Algorithms and Data Structures for Data Science
Trees 1

CS 277
Brad Solomon

October 11, 2021
Informal Early Feedback

Form available on Moodle (along with lab surveys)

Feedback form open until October 13th

If >50% class participates, everyone gets 10 bonus points
Learning Objectives

Formally define the tree data structure

Explore properties of trees and the specifics of binary trees

Implement and understand traversals and search on trees
Trees

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

(In CS 277) a tree is also:

1) Acyclic

2) Rooted
Tree Terminology

**Node:** The vertex of a tree

**Edge:** The [theoretical] connecting path between nodes

**Path:** A list of the edges (or nodes) traversed to go from node *start* to node *end*
Tree Terminology

**Parent:** The precursor node to the current node is the ‘parent’

**Child:** The nodes linked by the current node are it’s ‘children’

**Neighbor:** Parent or child

**Degree:** The number of children for a given node
**Tree Terminology**

**Root**: The start of a tree (the only node with no parent).

**Leaf**: The terminating nodes of a tree (have no children)

**Internal**: A node with at least one child
Tree ADT

**Insert:** Add an object into tree

**Remove:** Remove a specific object from tree

**Traverse:** Visit every node in tree (all objects)
There are many *types* of trees.
A **binary tree** is a tree $T$ such that:

$$T = \text{None}$$

or

$$T = \text{TreeNode}(\text{val}, T_L, T_R)$$
class treeNode:
    def __init__(self, val, ______, ______):
        self.val = val

    ______________________

    ______________________

class binaryTree:
    def __init__(self, root=None):
        self.root = root
Tree Terminology

**Height**: the length of the longest path from the root to a leaf
Tree Height

\[ T.\text{height}() = \]

Base Case:

Recursive Step:

Combining:
Defining a tree

tn1 = treeNode(1)
tn2 = treeNode(2)
tn3 = treeNode(3)
tn4 = treeNode(4)
tn5 = treeNode(5, tn1, tn2)
tn6 = treeNode(6, tn3, tn4)
tn7 = treeNode(7, tn5, tn6)
Tree Traversals
class binaryTree:

    def preorderTraversal(self):
        self.preorderHelper(self.root)

    def preorderHelper(self, node):
        if node:
            print(node.val)
            self.preorderHelper(node.left)
            self.preorderHelper(node.right)
Post-Order Traversal

```python
class binaryTree:
    def postorderTraversal(self):
        self.postorderHelper(self.root)
    def postorderHelper(self, node):
        if node:
            self.postorderHelper(node.left)
            self.postorderHelper(node.right)
            print(node.val)
```

```
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

```
1 ———— 5 ———— 7
|          |          |
|          |          |
|          |          |
|          |          |
```

```
2 ————       ———— 6
|          |          |
|          |          |
|          |          |
```

```
3 ———— 4
|          |
```

```
1 ———— 2 ———— 3 ———— 4
```
In-Order Traversal

class binaryTree:
    def inorderTraversal(self):
        self.inorderHelper(self.root)
    
def inorderHelper(self, node):
        if node:
            self.inorderHelper(node.left)
            print(node.val)
            self.inorderHelper(node.right)
Tree Traversals

Pre-order:

In-order:

Post-order:
Tree Traversals

**Pre-order**: Ideal for copying trees

**Post-order**: Ideal for deleting trees
Search Running Times on a Binary Tree
Traversal vs Search

Traversal:

Search:
Search: Breadth First vs. Depth First

Strategy: Depth First Search (DFS)
Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS)
Level-Order Traversal

Level-order:
Level-Order Traversal

class binaryTree:

    def levelorderTraversal(self):
        queue = []
        queue.append(self.root)
        while queue:
            curr = queue.pop(0)
            print(curr.val)
            if curr.left:
                queue.append(curr.left)
            if curr.right:
                queue.append(curr.right)
Search: Breadth First vs. Depth First

**Strategy:** Breadth First Search (BFS)

**Strategy:** Depth First Search (DFS)
What search algorithm is best?

The average ‘branch factor’ for a game of chess is ~31. If you were searching a decision tree for chess, which search algorithm would you use?
Improved search on a binary tree

5 3 6 7 1 4

1 3 4 5 6 7

5

3 6

7 1 4

5

3

1 4

6

7
Binary Search Tree (BST)

A BST is a binary tree $T = \text{treeNode}(\text{val}, T_L, T_r)$ such that:

$\forall n \in T_L, \ n\.\text{val} < T\.\text{val}$

$\forall n \in T_R, \ n\.\text{val} > T\.\text{val}$