Algorithms and Data Structures for Data Science

Trees

CS 277
Brad Solomon

March 20, 2023

Department of Computer Science
Assignment Extensions

Everyone will have until Wednesday (3/22) to complete lab_search

Everyone will have until Friday (3/24) to complete mp_hash
Informal Early Feedback

I feel that I can actively participate…

in lecture:
in class in general:

90.5% / 81% of class receive helpful and complete answers in / out of class.
Informal Early Feedback

Helpful for learning material…

Lectures:

Assignments:

- 1 - Not at all helpful
- 2
- 3
- 4
- 5 - Very helpful
Informal Early Feedback

Pacing of the course:

- 66.7% rated it as "Just right"
- 33.3% rated it as "Too Fast"
Informal Early Feedback

Labs and lectures are ‘most helpful’ to most students; MPs not listed at all

Requests / Suggestions (in no particular order):

Extending deadline for labs (or opening labs earlier in week)

Providing annotated lecture slides after each lecture

Include more live coding demonstrations in the class

Giving optional coding questions for practice exercises

Providing practice exams / example exam questions

A final project instead of a final
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
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 its ‘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
There are many types of trees
Binary Tree

A **binary tree** is a tree $T$ such that:

$$T = \text{None}$$

or

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

        __________________________
        __________________________
        __________________________

class binaryTree:
    def __init__(self, root=None):
        self.root = root
Defining a tree

class treeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

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)

binaryTree(___________)
Tree Terminology

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

What is the height of a tree with zero nodes?
Tree Height

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

**Base Case:**

**Recursive Step:**

**Combining:**
Tree Traversals

A **traversal** of a tree $T$ is an ordered way of visiting every node once.
Pre-Order Traversal

1) Get current node’s value
2) Recurse left
3) Recurse right
Post-Order Traversal

1) Recurse left
2) Recurse right
3) Get current nodes value
In-Order Traversal

1) Recurse left
2) Get current nodes value
3) Recurse right
Tree Traversals

Pre-order:

In-order:

Post-order:
Tree Traversals

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

**Post-order:** Ideal for deleting trees
Searching a Binary Tree

There are two main approaches to searching a binary tree:
Depth First Search

Explore as far along one path as possible before backtracking
Breadth First Search

Fully explore depth i before exploring depth i+1
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
Binary Search Tree (BST)

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

\[ \forall n \in T_L, \ n \cdot \text{val} < T \cdot \text{val} \]

\[ \forall n \in T_R, \ n \cdot \text{val} > T \cdot \text{val} \]