CS234
MODULE 10 – TREES

• What’s a tree
• Tree terminology
• Tree uses
• Tree algorithms

Updated: 2018-06-26
Trees

- Remember these?
- It’s a “binary tree”
- Like a linked list
- Links go left and right
- Can have trees with more than 2 links
Binary Tree Node (Python)

class BTNode:
    '''Fields: item – Any
                left,right – (anyof None
                            BTNode)'''

    def __init__(self,item):
        self.item = item
        self.left = self.right = None

    def __init__(self,item):
        self.item = item
        self.left = self.right = None
Tree Terminology

- Node has links to **children**
  - In a **binary tree**, they are called “left” and “right”
- Every node except **one** has a **parent**
  - (If A is B’s parent, B is A’s child)
- The **one** node without a parent is the **root**
- A node with no children is a **leaf**
- A node with children is **internal**
Node Terminology
**Height**

Height of a node – How far it is from being a leaf

- Height(\text{None}) = 0
- Height(\text{Node}) = 1 + \max(\text{Height(\text{Node.Left})}, \text{Height(\text{Node.Right})})

That’s for a binary tree (only left and right child). You can extend it for any tree, though.
Node Heights

![Tree Diagram]

- **root**: 8
- **Level 1**: 4 (1), 16 (3)
- **Level 2**: 15 (1), 23 (2), 42 (1)

Node heights are indicated as numbers next to each node.
Paths

A path in a tree is a sequence of nodes, where the next node is always a child of the current node.

Paths from root:
[8]
[8,4]
[8,16]
[8,16,15]
[8,16,23]
[8,16,23,42]
Paths

• The path is the list of nodes not their labels. As long as each label is unique the difference won’t matter.

• In fact, all you need is labels to be unique among siblings:
  • [8,8,8] is unambiguous
  • [8,8] and [8,4] are not

(They are fine if you know which 8 is the start)
Height and Path

Height of node N
- The longest path that originates at N

Height of tree T
- The longest path from the root of T
- Or
- The longest path in T
It’s all relative

• Ancestor: Parent node, Parent’s parent node, Parent’s parent’s parent node, etc.
  • The root is the ancestor of all nodes

• Descendant. Children, children’s children, etc.

• Siblings: The other children of your parent

• Cousins: Sure

• Niblings (nieces and nephews): Why not?
Relative Terms

7’s descendants: [2, 16, 3, 10, 4]
3’s siblings: [4]
2’s cousins: [3, 4]
2’s ancestors: [16, 7]
10’s niblings: [2]
3’s grandparent: 7
3’s aunt / uncle: 16

It can be difficult to determine the gender of a node
What Good is a Tree?

It organizes data according to parent-child relationships

Lets you establish some kind of hierarchy for your data

Matches real world things sometimes
File Systems are Trees

• Just a small part of my home file system
Flip it turnwise

Directories are easier to read if you turn the tree sideways

~djholty
  teaching
  cs136
    slides
    assns
    mt
  cs234
    slides
    assns
    mt
Oops, more terminology

Every node in a tree is the root of its own subtree.

My user directory is not the root of the file system, but it’s still a root. Each directory is a subtree (a subdirectory).

The path of a file is the same thing as a path in a tree.

/home/djholty/proj ect/LoopWeaver
Documents are Trees, too

Thesis.doc
Title Page
Abstract
Acknowledgements
Chapter: Introduction

Section: Proteins and Structure

Paragraph...

Figure...

...

Sub Section: Primary Structure

Paragraph...
General Tree Nodes

class TreeNode:
    def __init__(self, item):
        self.item = item
        self.children = []

    def add_child(self, child):
        assert isinstance(child, TreeNode)
        self.children.append(child)
General Tree Example

Root = TreeNode(1)
Root.add_child(TreeNode(2))
Root.add_child(TreeNode(3))
C = TreeNode(4)
Root.add_child(C)
C.add_child(TreeNode(5))
Back to Binary

There are some fun properties of binary trees:

If \( n \) is the number of nodes in a binary tree then

\[
  n \geq h \geq \lceil \log_2 (n + 1) \rceil
\]

(The “ceiling” isn’t really needed. \( h \geq 3.4 \) implies \( h \geq 4 \), since \( h \) must be an integer value).
The two extremes when n=7

Left: \( h = n = 7 \)

Right: \( h = \lg(n+1) = 3 \)
We can Prove it!

TL;DR: Each layer (counting from 0) contains at most $2^i$ nodes.

\[ n \leq 1 + 2 + 4 + 2^{h-1} = 2^h - 1 \]
\[ n+1 \leq 2^h \]
\[ \log_2 (n+1) \leq h \]
Traversing a Tree

There comes a time in every programmer’s life when they have to traverse a tree.

There are three fun and exciting ways to traverse a binary tree (and one other way).

The difference is what order you “process” the nodes in.
Pre-order Traversal

Process a node **before** its children, and process the children left to right.

Processing order: 1, 2, 4, 5, 3, 6, 7
Pre-order Traversal

Use a stack to keep track of what’s in progress.

The **call stack** will work (i.e. use recursion)

def preorder(node):
    if node is not None:
        process(node)
        preorder(node.left)
        preorder(node.right)
Post-order Traversal

Process a node **after** its children have been processed.

Processing order: 4,5,2,6,7,3,1
Post-order Traversal

It didn’t even start at the root. How?

It does start at the root, it just waits to process it until the end. Sounds stack-like again.

def postorder(node):
    if node is not None:
        postorder(node.left)
        postorder(node.right)
        process(node)
Traversing a General Tree

Pre-order and post-order traversals will work on a general tree, too.

The recursive calls would just look like:

```python
for child in node.children:
    recursive_call(child)
```
In-order Traversal

For a binary tree: process the node after its left child but before its right child.

Order: 4, 2, 5, 1, 6, 3, 7
In-order Traversal

Surprise: Recursion

def inorder(node):
    if node is not None:
        inorder(node.left)
        process(node)
        inorder(node.right)
Tree Application: Binary Expressions

We can use a binary tree to represent expressions:

class OpNode:
    '''Fields:  op – string (operator)
        left – ExpTree (operand)
        right – ExpTree (operand)'''

    ...

An ExpTree is either an operator (OpNode) or a value (Int or Float)
Binary Expression Trees

2+2

\[
\begin{array}{c}
+ \\
| \\
2 \\
| \\
2 \\
\end{array}
\]

2+3*4

\[
\begin{array}{c}
+ \\
| \\
2 \\
| \\
* \\
| \\
3 \\
| \\
4 \\
\end{array}
\]
def bt_eval(bet):
    if not isinstance(bet, OpNode):
        return bet # it’s an int or a float
    lhs = bt_eval(bet.left)
    rhs = bt_eval(bet.right)
    if bet.op == '+': return lhs + rhs
    elif bst.op == '-': return lhs – rhs
    ... etc
Binary Expression Trees - Printing

Pre-, Post-, and In-Order traversals will print: Prefix Notation, Postfix Notation, and Infix Notation.

(For infix notation we must use parentheses)
Binary Expression Trees - Printing

pre: + 2 2
post: 2 2 +
in: (2 + 2)

pre: + 2 * 3 4
post: 2 3 4 * +
in: (2 + (3 * 4))
Why Parentheses?

In-Order (No Parens)

$$2 + 3 \times 4$$

$$\text{In-Order (No Parens)}$$

With Parens

$$2 + (3 \times 4)$$

$$2 + 3 \times 4$$

$$((2 + 3) \times 4)$$
You Mentioned One More Traversal?

Did I? Oh yeah, the weird one.

All of the *-order traversals are called “depth-first” because you will reach a leaf (a “deep” node) before you finish with the root and its children.

We can also do a “breadth-first” search.
Breadth-First

First process the root, then all nodes that are 1 step away from the root, then all nodes that are 2 steps away, etc.

Order: 1,2,3,4,5,6,7
Breadth-First

def breadth(root):
    nodeQ = Queue()
    nodeQ.enqueue(root)
    while not nodeQ.empty():
        node = nodeQ.dequeue()
        process(node)
        if node.left:
            nodeQ.enqueue(node.left)
        if node.right:
            nodeQ.enqueue(node.right)
Why?

Breadth-First is useful for searches where you want to find the first match according to depth (the match that is closest to the root).

It’s also used for printing trees because: you want to print the root on one line, then all its children below, then all its grand children, etc.
Examples

• Dan will do these in class
• That means they’re probably posted
  • If not by Friday, bug remind me