Exam 1 next week

Multiple Choice / Fill in the blank exam

Covers content through Monday February 19th

See website for details

- Python Fundamentals
- Big O
- Lists
- Stack & Queue
Learning Objectives

Introduce recursion in the context of trees

Explore recursion in the context of loops

Practice recursion in the context of lists

Back to trees next week

U ADT → Implement (review) → Big O
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
   - No cycles in edges
   - No path from node to itself

2) Rooted
   - We have some labeled root node
   - Every node in tree can be reached by path from root
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

\[ C \rightarrow B \rightarrow E \rightarrow F \]

neighbor (C, B) (E, F) (F, E)
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

At most degree 2
**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
What is the longest path in the tree?

C - B - D - E

What is the neighbors of node B?

C, F, D, A

How many leaves does this tree have?

F, E, A, G are leaves (4)

What is the largest degree in the tree?

3 (Node B)
Tree Terminology

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

How does a *program* identify the height of a tree?

\[
\text{Height (root = A)} = 1 + H(B) \text{ or } H(C)
\]
Tree Height Calculation Breakdown

How does a program identify the height of a tree?

The height of my tree is 1 plus the height of my children!

To get $H(A)$

I need $H(B)$ and $H(C)$

Leaf has height = 0
Tree of None nodes = -1

I need $H(D)$ and $H(E)$ and…
I need $H(F)$ and $H(G)$ and…
The process by which a function calls itself directly or indirectly is called recursion.

Don’t panic — we’ve already used it before!
Linked List Recursion

A **linked list** is a list $L$ such that:

$L = None$

or

$L = listNode(val, L_{next})$
(Binary) Tree Recursion

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

$T = \text{None}$

or

$T = \text{treeNode}(\text{val}, T_L, T_R)$

```python
class treeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```
Visualizing a binary tree

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

a = treeNode('a')
b = treeNode('b')
c = treeNode('c')
d = treeNode('d')
e = treeNode('e')
f = treeNode('f')
g = treeNode('g')

a.left = b
a.right = c
b.right = d
b.left = e
c.right = f
f.right = g
Visualizing a binary tree... recursively

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

a = treeNode('a')
b = treeNode('b')
c = treeNode('c')
d = treeNode('d')
e = treeNode('e')
f = treeNode('f')
g = treeNode('g')
a.left = b
b.right = c
b.left = e
c.right = f
f.right = g
```

```
a = treeNode('a', treeNode('b', treeNode('e'), treeNode('d')),
            treeNode('c', None, treeNode('f', None, treeNode('g'))))
```
At its core, recursion is nothing more than another way of writing loops:

```python
def recursiveFor(n):
    if n == 0:
        print(n)
        return
    recursiveFor(n-1)
    print(n)
```

for i in range(n+1):  
    print(i)
Let's deep dive into what's actually happening here:

```python
def recursiveFor(n):
    if n == 0:
        print(n)
        return
    recursiveFor(n-1)
    print(n)
```

```python
def recursiveFor(n):
    if n == 0:
        print(0)
        return
    print(n)
    recursiveFor(n-1)
    print(n)
```

```python
1 def recursiveFor(n):
2     if n == 0:
3         print(n)
4         return
5
6     recursiveFor(n-1)
7
8     print(n)
```

```python
1 def recursiveFor(n):
2     if n == 0:
3         print(0)
4         return
5
6     print(n)
7
8     recursiveFor(n-1)
```

- `recursiveFor(2)`
- `recursiveFor(1)`
- `recursiveFor(0)`
- `print(2)`
- `print(1)`
- `print(0)`
- `print(0)`
What is the following code doing?

```python
def recurse(i):
    if i == 0:
        return i
    return recurse(i-1)+i
```

```python
def recurse(inList):
    if len(inList)==0:
        return 0
    inList.pop()
    return recurse(inList)+1
```

1) Give a specific (small) example

@ line 4: return recurse(0)+1
Programming Toolbox: Recursion

Anything that can be solved with a loop can be solved with recursion

But sometimes it's easier to code up a solution recursively

I can't loop through a tree with `for` or `while`…

But I can loop through the tree using recursion!
Programming Toolbox: Recursion

When thinking recursively, break the problem into parts:

**Base Case:** What is the smallest sub-problem? What is the trivial solution?

When \( L = \text{length} \)

**Recursive Step:** How can I reduce my problem to an easier one?

\( i + 1 \)

**Combining:** How can I build my solution from recursive pieces?

\( \text{How to pass the value back} \)
Recursive Tree Height

What is the height of my tree $T$?

**Base Case:** What is the smallest sub-problem? What is the trivial solution?
- Tree is leaf (1 node case, no children) = height is 0
- Tree is empty (0 node case) = height is -1

**Recursive Step:** How can I reduce my problem to an easier one?
- $\text{Height}(T) = \max(\text{height}(T_{\text{left}}), \text{height}(T_{\text{right}}))$

**Combining:** How can I build my solution from recursive pieces?
Recursive Sum

Given a list, sum all the items in the list using recursion

**Base Case:** What is the smallest sub-problem? What is the trivial solution?
- A list of length 0 $\rightarrow$ return 0
- Length 1 $\rightarrow$ return the one item

**Recursive Step:** How can I reduce my problem to an easier one?
- Pop() or remove() one item (label it x)

A list of length n-1 is easier than a list of length n

**Combining:** How can I build my solution from recursive pieces?

- $\leftarrow$ return $x + \text{Recursive Sum}(\text{list of n-1 items})$
Recursive Sum

Given a list, sum all the items in the list using recursion

8 4 2 6 5
Recursive findMax

Given a list, find the max item in the list using recursion

Base Case:

Recursive Step:

Combining:
Recursive findMax

Given a list, find the max item in the list using recursion

8 4 2 6 5
Recursive Fibonacci

Given a number $n$, return the $n$th Fibonacci number:

$$Fib(n) = Fib(n - 1) + Fib(n - 2), \quad n > 1$$

Base Case:

Recursive Step:

Combining:
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

\[
\begin{array}{cccccc}
6 & 5 & 4 & 2 & 7 \\
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 3 & 3 & 3 & 3 & 1 \\
\end{array}
\]
Recursive List Partitioning

How would a computer solve this problem?

[6 5 4 2]
Recursive List Partitioning

How would a computer solve this problem? **Compute every permutation!**

6 5 4 2

6

6 5

6 5 4

6 2

5 4 2

4 2

2

5 4

...
Recursive List Partitioning

Writing code to attempt every possible permutation is tricky with loops.

But its a great example of recursion in action!

**Recursive Step:** Given list L, pop() L[0] to left *and* right and recurse on both
Recursive List Partitioning

**Recursive Step:** Given list L, pop(L[0]) to left *and* right and recurse on both

**Input:**

```
6 5 4 2
```

**Recursive Calls:**

```
5 4 2
```

```
5 4 2
```

**Left**

```

```

```
```

**Right**

```

```

```
```
Recursive List Partitioning

**Recursive Step:** Given list L, pop() L[0] to left and right and recurse on both

**Base Case:**
**Base Case:** When my input list is empty, I have tried every permutation

**Recursive Step:** Given list L, pop() L[0] to left and right and recurse on both

```
[4, 3, 1]  
([], [])
```

```
[3, 1]  
([4], [])  
([], [4])
```

```
[1]  
([3, 4], [])  
([4], [3])  
([3], [4])  
([], [3, 4])
```

```
[]  
([1, 3, 4], [])  
([1, 4], [3])  
([1, 3], [4])  
([1], [3, 4])  
([3, 4], [1])  
([4], [1, 3])  
([3], [1, 4])  
([], [1, 3, 4])
```
Recursive List Partitioning

**Base Case:** When my input list is empty, I have tried every permutation

**Recursive Step:** Given list L, pop() L[0] to left and right and recurse on both

**Combination Step:**
Lab Recursion

Recursive List Partitioning is now extra credit on Fridays lab!

In preparation for Friday, consider how you might use recursion to solve:

Computing the factorial of a number

Counting the sum of all digits in a number

Checking if a string is a palindrome