Trees and Human Evolution

orangutan  gorilla  human  common chimpanzee  bonobo

~15 MYA  ~7 MYA  ~6 MYA  ~3 MYA

~15 MYA
Learning Goals

- Introduce biological question
- Describe how data is stored in memory (just a peek)
- Describe tree terminology and representation
- Practice writing functions on trees
This week’s homework

- Write memoized version of fold
- Find regions of strong secondary structure in HIV pol gene
- Find pairing interactions
- Drawing secondary structure interactions
- Add steric constraints!

```
>>> shortestPath("A", "E", FiveCities, FiveDists)
10
```

```
[10, ['A', 'C', 'D', 'E']]
```

“Care-package” -ization
Coming Soon to CS 5 Green

- Phylogenetics
- OOPs
- CS Theory
- End-of-semester projects!

What we are ANTicipating…
Neanderthals and Modern Humans

Neanderthal type specimen

the old man of La Chapelle

https://www.msu.edu/~heslipst/contents/ANP440/images/Neanderthal_1_langle.jpg
http://humanorigins.si.edu/evidence/human-fossils/fossils/la-chapelle-aux-saints
http://anthropologynet.files.wordpress.com/2007/06/neander-valley.jpg
Trees and Human Evolution

orangutan  gorilla  human  common chimpanzee  bonobo

~15 MYA  ~7 MYA  ~6 MYA  ~3 MYA
Homo erectus: first undisputed world traveler

Sangiran 17, 1.3-1.0 MYA, Sangiran Indonesia
Out of Africa vs. multiregional origin of modern humans

- **H. erectus** in Africa
- **H. erectus** in Europe
- **H. erectus** in Asia

- **H. sapiens** in Africa
- **H. sapiens** in Europe
- **H. sapiens** in Asia

Intermediate species possible
Differing predictions...

Multiregional model

Out of Africa model
Learning Goals

• Introduce biological question
• Describe how data is stored in memory (just a peek)
• Describe tree terminology and representation
• Practice writing functions on trees
Lists Revisited

>>> L = [4, 5, 7]

>>> M = L

>>> N = L + [9]

This is called “shallow copy”
>>> L = [4, 5, 7]
>>> M = L
>>> N = L + [9]

>>> L.append(6)

>>> L
[4, 5, 7, 6]

>>> M
[4, 5, 7]

>>> N
[4, 5, 7, 9]
>>> L = [4, 5, 7]
>>> M = L
>>> N = L + [9]

>>> L.append(6)
>>> L  
[4, 5, 7, 6]
>>> M  
[4, 5, 7, 6]
>>> N  
[4, 5, 7, 9]
Strings are **immutable**

```python
>>> F = "Spam"

>>> F.append("!")
BARF! F.append("!")
AttributeError: 'str' object has no attribute 'append'
```

Python should really just say “BARF!”
Tuples are immutable lists

```python
>>> T = (4, 5, 7)
>>> T[0]
4
>>> T[1:]
(5, 7)
>>> len(T)
3
>>> T[0] = 42
BARF!
>>> T.append(42)
BARF!
```
Tuples are immutable lists

```python
>>> T = (4, 5, 7)
>>> U = T + (11, 12)
>>> U
(4, 5, 7, 11, 12)
>>> V = T + (5)
 ← something weird happens here!
BARF!
>>> V = T + (5,)
>>> V
(4, 5, 7, 5)
```
Learning Goals

- Introduce biological question
- Describe how data is stored in memory (just a peek)
- Describe tree terminology and representation
- Practice writing functions on trees
What about the Sontag, South, East, and Atwood Groodies?

Phylogenetic Trees

these are called leaf nodes or leaves or tips of the tree

How do we represent this in Python?
R LR (root, left, right) format

groodies = ("X",
            ("Y",
              ("W", (), ()),
              ("Z",
                ("E", (), ()),
                ("L", (), ()))),
            ("C", (), ()))
tr = ('Q',
     ('R',
      ('T',
       ('V', ()), ()),
       ('Z', ()), ()),
     ('S',
      ('X', ()), ()),
      ('W', ()), ()),
     ('Y', ()))
Draw this tree...

\[
\text{tr} = ('Q',
    ('R',
        ('T',
            ('V', (), ()),
            ('Z', (), ()),
        ),
        ('S',
            ('X', (), ()),
            ('W', (), ()),
        ),
    ),
    ('Y', (), ()),
)
\]
Learning Goals
• Introduce biological question
• Describe how data is stored in memory (just a peek)
• Describe tree terminology and representation
• Practice writing functions on trees
Traversing Trees

Print name of every node in the tree so that parents always appear...

- *before* children
  - *preorder* traversal
  - e.g. Anc0, dog, Anc1, human, Anc2, mouse, rat

- *after* children
  - *postorder* traversal
  - e.g. dog, human, mouse, rat, Anc2, Anc1, Anc0
Traversing Trees

Preorder (parents first)

```python
def preorder_print(tree):
    root, left, right = tree
```

Postorder (children first)

```python
def postorder_print(tree):
    root, left, right = tree
```

Use recursion. Start with base case!
How many nodes are in this tree?

```python
def node_count(tree):
    """Returns the total number of nodes in the given tree."""

>>> node_count(('Yoohoo', (), ()))
1
>>> node_count(groodies)
7
```

Fill this in (in your notes)!
How many nodes are in this tree?

>>> node_count(('Yoohoo', (), ()))
1
>>> node_count(groodies)
7

You should always assume that if one child is () then so is the other!

def node_count(tree):
    """Returns the total number of nodes in the given tree."""
    root, left, right = tree # root = tree[0], left = tree[1], right = tree[2]
    if left == (): return 1  # a leaf
    else:                    # an internal node
        return 1 + node_count(left) + node_count(right)

Fill this in (in your notes)!

It would be a shame to "leaf" out the base case!
def find(species, tree):
    """Returns True if species is in tree and False otherwise."""
    root, left, right = tree
    if root == species: return True  # found it at the root!
>>> find("E", groodies)
True
>>> find("Sontag", groodies)
False

def find(species, tree):
    """Returns True if species is in tree and False otherwise."""
    root, left, right = tree
    if root == species: return True  # found it at the root!
    if left == (): return False
    else:
        return find(species, left) or find(species, right)
The height of a tree is the length of the path from the root to the deepest node in the tree.

```python
def height(tree):
    """Returns the height of the given Tree."""
    root, left, right = tree
    return max(height(left), height(right)) + 1

>>> height(groodies)
3
>>> height(('spam', (), ()))
0
```

Worksheet
The height of a tree is the length of the path from the root to the deepest node in the tree.

```python
def height(tree):
    """Returns the height of the given Tree."""
    root, left, right = tree
    if left == (): return 0  # a leaf
    else:  # an internal node
        return 1 + max(height(left), height(right))
```

```python
>>> height(groodies)
3
>>> height(('spam', (), ()))
0
```

Worksheet

West Dorm Groody ("W")
East Dorm Groody ("E")
Linde Dorm Groody ("L")
Case Groody ("C")
node_list

```python
def node_list(tree):
    """Returns the list of nodes in a given tree."""
    root, left, right = tree
```

```python
>>> node_list(groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']
```

```
X
```

```
Y
    Z
```

West Dorm Groody ("W")
East Dorm Groody ("E")
Linde Dorm Groody ("L")
Case Groody ("C")
```python
def node_list(tree):
    """Returns the list of nodes in a given tree."""
    root, left, right = tree
    if left == (): return [root]
    else:
        return [root] + node_list(left) + node_list(right)
```
def leaf_list(tree):
    """Returns the list of leaves in a given Tree."""
    root, left, right = tree

>>> leaf_list(groodies)
['W', 'E', 'L', 'C']
def leaf_list(tree):
    """Returns the list of leaves in a given Tree."""
    root, left, right = tree
    if left == ():  return [root]
    else:
        return leaf_list(left) + leaf_list(right)

>>> leaf_list(groodies)
['W', 'E', 'L', 'C']
Isomorphic Trees

Consider three trees

T1

("Anc",
("W", (), ()),
("Anc",
("L", (), ()),
("E", (), ()))
)

W

L

E

T2

("Anc",
("Anc",
("E", (), ()),
("L", (), ()))
,
("W", (), ()))
)

E

L

W

T3

("Anc",
("L", (), ()),
("Anc",
("E", (), ()),
("W", (), ()))
)

L

W

E

Two trees are isomorphic if they imply the same underlying phylogenetic relationships

• T1 and T2 are isomorphic
• T3 is not isomorphic either with T1 or T2
Isomorphic Trees

```python
def isomorphic(tree1, tree2):
    """Returns boolean indicating if tree1 and tree2 are isomorphic."""

>>> isomorphic(('A',(),()),('A',(),()))
True
>>> isomorphic(('A',(),()),T1)
False
>>> isomorphic(('A',(),()),('B',(),()))
False
>>> isomorphic(T1,T2)
True
>>> isomorphic(T1,T3)
False
>>> isomorphic(T2,T3)
False
```
Isomorphic Trees

```python
def isomorphic(tree1, tree2):
    """Returns boolean indicating if tree1 and tree2 are isomorphic."""
    if tree1 == tree2: return True
    elif tree1[1] == () or tree2[1] == (): return False
    else:
        option1 = isomorphic(tree1[1], tree2[1]) and \
                  isomorphic(tree1[2], tree2[2])
        option2 = isomorphic(tree1[1], tree2[2]) and \
                  isomorphic(tree1[2], tree2[1])
    return option1 or option2
```

```python
>>> isomorphic(('A',(),()),('A',(),()))
True
>>> isomorphic(('A',(),()),T1)
False
>>> isomorphic(('A',(),()),('B',(),()))
False
>>> isomorphic(T1,T2)
True
>>> isomorphic(T1,T3)
False
>>> isomorphic(T2,T3)
False
```
Traversing Trees

Preorder (parents first)

```python
def preorder_print(tree):
    root, left, right = tree
```

Postorder (children first)

```python
def postorder_print(tree):
    root, left, right = tree
```
Traversing Trees

Preorder (parents first)

```python
def preorder_print(tree):
    root, left, right = tree
    if left == ():
        print(root)
    else:
        preorder_print(left)
        preorder_print(right)
```

Postorder (children first)

```python
def postorder_print(tree):
    root, left, right = tree
    if left == ():
        print(root)
    else:
        postorder_print(left)
        postorder_print(right)
        print(root)
```

Use recursion. Start with base case!