Entertainment: CS 5 Green book to be made into feature-length movie starring George Clooney as the happy turtle.

News in Brief

HMC CS Department to replace Apple computers with new Pumpkin brand computers. (p. 42)

Talking tree debuts in CS 5 Green!

New computer program predicts today’s weather: IndexError: list index out of range

Sports: CS 5 Green Prof runs to class in record time. Were steroids involved?

CS 5 Green Today

HMC CS 5 Green Professors discover new discovery

Claremont, CA: Researchers at Harvey Mudd College have made an extremely important new discovery said a spokesperson for the College. The discovery was evidently discovered while the researchers were trying to discover another discovery. “The professors discovered that their discovery had not been previously discovered, which is an important discovery in its own right,” said the excited spokesperson. A number of prominent scientists also expressed their tremendous enthusiasm and said that they looked forward to reading what was actually discovered.
Phylogenetic Trees...

From Darwin’s notebooks, 1837
I think
Really?
The only figure in *On the Origin of Species by Natural Selection* (Darwin, 1859)

From the 6th edition (1872): “The affinities of all the beings of the same class have sometimes been represented by a great tree. I believe this simile largely speaks the truth. The green and budding twigs may represent existing species; and those produced during former years may represent the long succession of extinct species.”
More recursion on trees: mrca

>>> find("L", Groodies)
True

>>> mrca("L", "E", Groodies)  # use find as a helper!
'Z'
>>> mrca("W", "E", Groodies)
'Y'
>>> mrca("W", "C", Groodies)
'X'
>>> mrca("W", "Prof. Wu", Groodies)
None
def mrca(species1, species2, Tree):
    """Return the name of the most recent common ancestor of species1 and species2. If there isn't one, return None."""
def mrca(species1, species2, Tree):
    """Return the name of the most recent common ancestor of species1 and species2. If there isn't one, return None."""
    if Tree[1] == (): return None
    elif not find(species1, Tree) or not find(species2, Tree):
        return None
    else:
        if find(species1, Tree[1]) and find(species2, Tree[1]):
            return mrca(species1, species2, Tree[1])
        elif find(species1, Tree[2]) and find(species2, Tree[2]):
            return mrca(species1, species2, Tree[2])
        else:
            return Tree[0]
**Input:** DNA or protein sequences for each species

**Output:** Species tree

Distance-Based Approach

Groody  CATCAACCAGTGACCAGTATAGGACGCCC
Froody  CAACACTCAGTGACAAGTCTAGCAGGCC
Snoody  AATCGCGCCGGCCTCAGGCATAGCTAGCGC
Distance-Based Approach

<table>
<thead>
<tr>
<th>G</th>
<th>F</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Current time ("time 0")

```
(6.0,
 ('Snoody', (), ()),
 (3.0,
  ('Groody', (), ()),
  ('Froody', (), ()))
)
```
Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>F</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>S</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>
Try This One…

The matrix is symmetric, so we just need to keep the bottom or top half!

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

This algorithm’s “bark” is worse than it’s bite.

The matrix is symmetric, so we just need to keep the bottom or top half!
I wood love to see this in action!

DEMO!
Implementing UPGMA

1 Groody
2 Froody
3 Snoody

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \text{groodiesList} = [N1, N2, N3] \]

Let’s see this in the provided mitoData.py file
N1 = ("Groody", (), ())
N2 = ("Froody", (), ())
N3 = ("Snoody", (), ())
groodiesList = [N1, N2, N3]
groodiesMatrix = {
    (N1, N1):0, (N1, N2):6, (N1, N3):12,
    (N2, N1):6, (N2, N2):0, (N2, N3):12,
}

1. For all pairs of trees in the groodiesList, find the closest pair
1. For all pairs of trees in the groodiesList, find the closest pair

N1 = ("Groody", (), ()
N2 = ("Froody", (), ()
N3 = ("Snoody", (), ()
groodiesList = [N1, N2, N3]
groodiesMatrix = {(N1, N1):0, (N1, N2):6, (N1, N3):12,
                   (N2, N1):6, (N2, N2):0, (N2, N3):12,
                   (N3,N1):12, (N3,N2):12, (N3,N3):0}
1. For all pairs of trees in the groodiesList, find the closest pair

N1 = ("Groody", (), ())
N2 = ("Froody", (), ())
N3 = ("Snoody", (), ())
groodiesList = [N1, N2, N3]
groodiesMatrix = {(N1, N1):0, (N1, N2):6, (N1, N3):12,
(N2, N1):6, (N2, N2):0, (N2, N3):12,
(N3, N1):12, (N3, N2):12, (N3, N3):0}

2. Remove those from the groodiesList

groodyList.remove(N1)
groodyList.remove(N2)
N1 = ("Groody", (), ())
N2 = ("Froody", (), ())
N3 = ("Snoody", (), ())
groodiesList = [N1, N2, N3]
groodiesMatrix = {(N1, N1):0, (N1, N2):6, (N1, N3):12,
                 (N2, N1):6, (N2, N2):0, (N2, N3):12,
                 (N3,N1):12, (N3,N2):12, (N3,N3):0}

1. For all pairs of trees in the groodiesList, find the closest pair

N1 = ("Groody", (), ())
N2 = ("Froody", (), ())

2. Remove those from the groodiesList

3. Make a new tree by joining these two trees...
1. For all pairs of trees in the groodiesList, find the closest pair

\[
N_1 = ("Groody", (), ()),
N_2 = ("Froody", (), ()),
N_3 = ("Snoody", (), ())
\]

groodiesList = [N_1, N_2, N_3]
groodiesMatrix = {
    (N_1, N_1):0, (N_1, N_2):6, (N_1, N_3):12,
    (N_2, N_1):6, (N_2, N_2):0, (N_2, N_3):12,
}

2. Remove those from the groodiesList

3. Make a new tree by joining these two trees...

\[
(3.0, N_1, N_2) =
(3.0, ("Groody", (), ()), ("Froody", (), ()))
\]
1. For all *pairs* of trees in the `groodiesList`, find the closest pair

\[
N1 = ("Groody", (), ()), \quad N2 = ("Froody", (), ()), \quad N3 = ("Snoody", (), ()),
\]

\[
groodiesList = [N3, (3.0, ("Groody", (), ()), ("Froody", (), ()))]

2. Remove those from the `groodiesList`

3. Make a new tree by joining these two trees...

\[
(3.0, N1, N2) = (3.0, ("Groody", (), ()), ("Froody", (), ()))
\]

4. Add this new tree to the `groodiesList`
N1 = ("Groody", (), ())
N2 = ("Froody", (), ())
N3 = ("Snoody", (), ())
groodiesList = [N3, (3.0, ("Groody", (), ()), ("Froody", (), ()))]
groodiesMatrix = {
    (N1, N1): 0, (N1, N2): 6, (N1, N3): 12,
    (N2, N1): 6, (N2, N2): 0, (N2, N3): 12,
}

5. Update the distance matrix...

We are still holding on to N1 and N2, even though we have removed them from the groodiesList!
Implementing UPGMA

findClosestPair( speciesList, Distances ):

""" Takes a list of species trees and the distance dictionary as input and returns a tuple (X, Y) where X and Y are in the list and have the minimum distance between any two items in the list. """

updateDist( speciesList, Distances, newTree):

""" Takes a list of species trees, the distance dictionary, and a newTree that was just formed by merging two trees found by findClosestPair. Those two trees can be found by looking inside newTree. Those two trees are removed from the distance dictionary and the newTree is added to the dictionary. """

upgma( speciesList, Distances):

""" returns the phylogenetic tree constructed by the UPGMA algorithm """
Updating the distance dictionary: a subtlety

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>12</td>
<td>12</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a,b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a,b</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a,b,c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a,b,c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- a
  - b
    - c
      - d

\[ \text{Distance: } 3 \]

- c
  - a
    - b

\[ \text{Distance: } 2 \]
### Updating the distance dictionary: a subtlety

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>12</td>
<td>12</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

When we combine distances between nodes with different numbers of leaves, we should weight by the number of leaves.

When we combine distances between nodes with different numbers of leaves, we should weight by the number of leaves.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td>a,b</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

2 leaves: $\frac{2}{3} \times 12 + \frac{1}{3} \times 9 = 11$

1 leaf: $\frac{1}{3}$

Total: $3$

When we combine distances between nodes with different numbers of leaves, we should weight by the number of leaves.
Updating the distance dictionary: a subtlety

When we combine distances between nodes with different numbers of leaves, we should weight by the number of leaves.

\[
\text{Distance(newTree, T3) = Distance(T1, T3) x leafCount(T1)/leafCount(newTree) + Distance(T2, T3) x leafCount(T2)/leafCount(newTree)}
\]

newTree = (“Anc”, T1, T2)
Inferring time...

The `scale` function you write in lab will be useful...
UPGMA assumes a molecular clock

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>F</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>S</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>
A note on biogeography/migrations