Points: 100

Neighbor Joining Tree Construction

Write a Python program `nj.py` based on the upgma code that we gave in class that will read in a lower triangular distance matrix and generate the groupings and lengths of edges of a non-rooted tree that "fits" the data. You must work from the supplied upgma.py program. Your program does not need to draw a tree. An machine readable copy of sample data from Felsenstien is here: ../dogbear.dat.

To illustrate the format of the input it looks like:

```
Dog Bear Raccoon Weasel Seal SeaLion Cat Monkey
  32
  48   26
  51   34   42
  50   29   44   44
  48   33   44   38   24
  98   84   92   86   89   90
 148  136  152  142  142  142  148
```

and is in the same form as for our upgma. Feel free to use the lib515.py library.

Output

For your information, here is the tree for the dogbear.dat data in Newick format with distances following each leaf and internal node:

```
((Bear:6.87500, Raccoon:19.12500):1.75000, ((Seal:12.35000,
  Sealion:11.65000):7.81250, (Weasel:19.56250, (Cat:47.08333,
  Monkey:100.91667):20.43750):1.56250):3.43750, Dog:25.25000);
```

However, your program needs to output exactly three lists:

- The list we called `merge` in the upgma example.
- The list we called `size` in the upgma example.
- A list of edge pairs (as tuples) for each internal node except the last. These are the distances to each of the subclusters in a cluster. The last element in the list is the final distance between the last two clusters to be merged.

```
[0, 1, 2, 3, 4, 5, 6, 7, (7, 6), (5, 4), (2, 1), (10, 0), (8, 3), (11,
  9), (12, 13)]
[1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 5, 8]
[0, 0, 0, 0, 0, 0, 0, 0, (100.91666666666666, 47.083333333333334),
  (11.650000000000006, 12.349999999999998), (19.125, 6.875), (1.75,
  25.25), (20.4375, 19.5625), (3.4375, 7.8125), 1.5625]
```

Notice that in order to get the correct output the final step must be to put the final distance on the list of edge pairs and the final cluster and size on their respective lists.

The Neighbor-joining Algorithm

Given a distance matrix d compute an unrooted tree topology complete with edge lengths that tries to preserve the additive property: $d_{i,m} + d_{j,m} - d_{i,j} = 2d_{k,m}$ where k is the first node on both routes from i and j to m.

1. Let the set of clusters be called L and initially $i \to C_i\ \forall i$ that is $|C_i| = 1$ and $L = C_1, C_2, \ldots C_N$.
2. $d_{i,j}$ is the distance from the initial distance matrix.
3. Compute "normalized distance matrix" $D_{i,j}$ for all i, j such that

   $$D_{i,j} = d_{i,j} - (r_i + r_j) \text{ where } r_i = \frac{1}{|L| - 2} \sum_{z \in L} d_{i,z}$$

   This subtracts the average distance to all other nodes than the pair involved. Note: this is **not** where we use the distance identity.
4. Use normalized distance to find $(i, j) = \underset{C_i, C_j \in L}{\operatorname{argmin}} D_{i,j}$
5. Merge $C_i \cup C_j \to C_k$ where k is a new cluster number.
6. Mark old clusters as used so that effectively: $L \leftarrow L - C_i - C_j$
7. Compute a new unnormalized distance matrix including the new cluster k and excluding i, j.

   $$d_{k,z} = d_{z,k} = \frac{1}{2}(d_{i,z} + d_{j,z} - d_{i,j}) \text{ for all } z \in L$$

   This uses the additivity of the distances to compute the distance to the new cluster from each other node.
8. Compute the length of the edges from k to i and j. Even though C_k has assumed the role of both C_i and C_j you still need the edge length to i and j from k in order to "draw" the tree.

   $$edge_{i,k} = \frac{1}{2}(d_{i,j} + r_i - r_j), \qquad edge_{j,k} = \frac{1}{2}(d_{i,j} + r_j - r_i)$$

9. Define height $h_k = d_{i,j}/2$ where h_k is the height of node that is the ancestor to all in C_k. When drawing the tree h_k is the height above the baseline (where all the leaves are).
10. $L \leftarrow L \cup C_k$
11. While there is more than two clusters left go to step 3
12. Finally, join the remaining two clusters with:

    $$edge_{j,k} = d_{i,j}$$

Implementation Notes

Consider this part of the computation:

$$D_{i,j} = d_{i,j} - (r_i + r_j) \text{ where } r_i = \frac{1}{|L| - 2} \sum_{z \in L} d_{i,z}$$

The values of r_z can be computed once each time we want to compute matrix D. This saves a vast amount of time. Furthermore, since $D_{i,j}$ is only used to find the argmin of $D_{i,j}$ we actually don't have to save array D; we only need to find the argmin of it. So first compute all the r and then combine the argmin step with the computation of $D_{i,j}$.

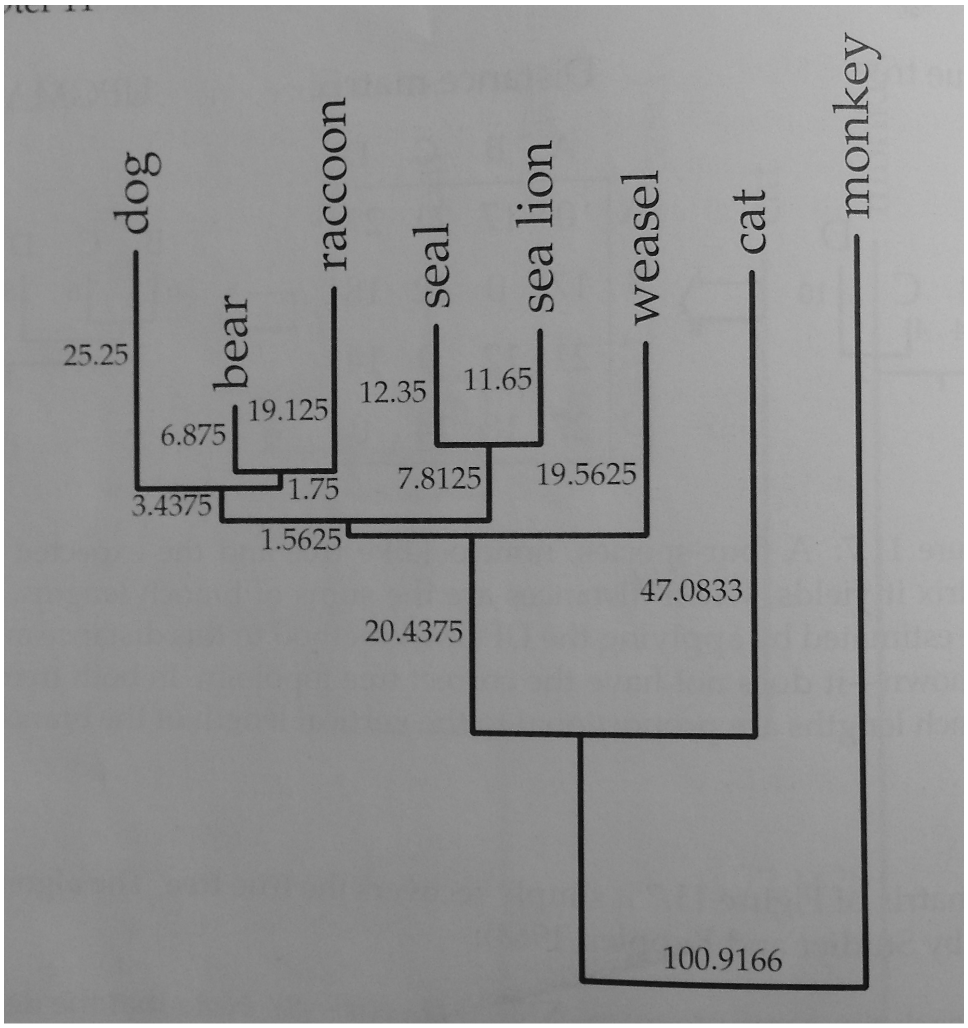

Figure 1: The non-rooted tree for the given problem drawn as a rooted tree.

In the box is an excerpt from the class notes on the algorithm.

Submission

As always, do your own work. Do not copy from others or from the internet. Using what we did in class is perfectly legal. Submit a single file named `nj.py` to the class submission page. The test script will ignore the spacing you choose and focus on non-whitespace. Remember no late papers and so always turn something in for partial credit.