Phylogenetic Trees

This problem brings together many different programming constructs and techniques we covered over the course of the semester including: string manipulation, (Python) lists, dictionaries, tuples, classes, list comprehensions, and trees. It is one of the most technically challenging programs assigned in this class this semester.

Background

An evolutionary tree (also called a phylogenetic tree) is a tree that expresses evolutionary relationships between a set of organisms.

This program involves writing code to construct phylogenetic trees starting from the genome sequences of a set of organisms. (Of course, there is nothing inherently genetic about the techniques we use and the code we write: for example, since programs are sequences of characters, we could just as easily apply this approach to sets of programs.)

Expected Behavior

Write a Python program, in a file **`phylo.py`**, that behaves as specified below.

1. *Read in the input parameters*:
    - Read in the name of an input file using **`input('FASTA file: ')`**.
    - Read in an integer value **`N`** using **`input('n-gram size: ')`**.

2. *Read in the input file*. The file format is specified under **Input format** (below).

    For each organism whose genome sequence appears in the input file:

    - Extract its ID (a string) from the first line of its information.
    - This first line will be followed by the organism's genome sequence, which is a long sequence of consisting of the letters `A`, `T`, `C`, and `G`. This sequence will appear over a number of adjacent lines and will be terminated by an empty line. Read in the genome sequence and concatenate the different lines into a single string.

3. For each organism, construct the set of its N-grams using the value **`N`** specified by the user in Step 1. [**Programming requirements**: item #3].)

    Note: You should construct a set of N-grams, i.e., an unordered collection without duplicates. If you don't, the similarity values you compute will be different from ours, and the phylogenetic tree will therefore also be different. You can learn more about Python sets here.

4. For each pair of distinct organisms *name1* and *name2* (i.e., *name1* ≠ *name2*), use the N-grams computed in the previous step to compute the *similarity* between their genome sequences (see **Algorithm A1** below). [**Programming requirements**: item #4].)

5. Use the similarity values computed in the previous step to compute the phylogenetic tree (see **Algorithm A2** and **Algorithm A3** below.) [**Programming requirements**: item #5].)

6. Print out the phylogenetic tree. Some examples are shown here.

Input format

The input file uses a text-based format called FASTA. An file can contain the genome sequences for several different organisms. Each such genome sequence has the following format:

1. A description line, which begins with the character '>' and contains identifying information about the genome sequence.

   For the purposes of this program, each genome sequence has an "ID" that is used to refer to it. This ID is obtained from the description line as follows: it begins at the character after the initial '>' and continues up to the first whitespace (blank or tab) character in the line.

2. The description line is followed by one or more non-empty lines of genome sequence data. **For the purposes of this program**, these lines should should be concatenated into a single string.

3. The genome sequence ends when an empty line is encountered.

For example, the following is a small portion of the Zika virus genome (source: www.ncbi.nlm.nih.gov): The ID for this genome sequence, shown highlighted in yellow, is `NC_012532.1`.

```
>NC_012532.1 Zika virus, complete genome
AGTTGTTGATCTGTGTGAGTCAGACTGCGACAGTTCGAGTCTGAAGCGAGAGCTAACAACAGTATCAACA
GGTTTAATTTGGATTTGGAAACGAGAGTTTCTGGTCATGAAAAACCCCAAAGAAGAAATCCGGAGGATCC
...
AGACTCCATGAGTTTCCACCACGCTGGCCGCCAGGCACAGATCGCCGAACTTCGGCGGCCGGTGTGGGGA
AATCCATGGTTTCT
```

Output format

The output format for a tree is as follows:

```
def __str__(self):
    if self.is_leaf():
        return self.id()
    else:
        return "({}, {})".format(str(self.left()), str(self.right()))
```

Here the method **`is_leaf()`** returns True if and only if the tree node it is invoked on is a leaf node; and **`id()`**, **`left()`**, and **`right()`** return, respectively, the ID of the node and the left and right subtrees of the tree node.

Some examples are shown here.

Pretty Printing for trees: If you want, you can use the code described at this link prettyprinting to print out your tree in a more visually appealing manner. However, this code is NOT part of the problem specification and is provided only to give you a better visualization of a tree. If you use this code, make sure you turn off the pretty-printing before you submit your code.

Algorithms

Algorithm A1. Similarity between two genome sequences

The similarity between two sequences is a number between 0 and 1: a similarity of 1 indicates that the sequences are identical, while a similarity of 0 indicates that the sequences have nothing in common. The intuition behind our algorithm for computing the similarity between two sequences is that if we chop each sequence up into small pieces, then similar sequences will have a lot of pieces in common while dissimilar sequences will not.

This intuition is formalized using the Jaccard index. Given two sequences *A* and *B*, let the corresponding sets of N-grams be *ngrams*(*A*) and *ngrams*(*B*) respectively. Then, the similarity between *A* and *B* is given by

$$similarity(A, B) = \frac{|\ ngrams(A) \cap ngrams(B)\ |}{|\ ngrams(A) \cup ngrams(B)\ |}$$

where |*S*| denotes the size of the set *S*. Note that different values of N for computing N-grams can produce different distance values for the same sequences, and thereby give rise to different phylogenetic trees.

For the purposes of this assignment, you should implement your N-grams as Python sets. The reason is to simplify the computation of unions and intersections:

- to create an empty set: **`set()`**
- the set corresponding to a list *L* is given by: **`set(`*`L`*`)`**
- the union of sets *S1* and *S2* is given by: **`S1.union(S2)`**
- the intersection of sets *S1* and *S2* is given by: **`S1.intersection(S2)`**

Note: Treating the N-grams for genome sequences as sets means that the presence or absence of duplicates in the sequences will not affect their similarity measure. This may not always seem intuitive.

Algorithm A2. Distance between two phylogenetic trees

Algorithm A1, discussed above, computes the similarity between two sequences. However, the tree construction process described below proceeds by computing the distance between *trees*, where each tree contains a set of sequences. So the sequence-similarity algorithm described above has to be extended to deal with sets of sequences. This can be done in a number of different ways, of which we chose one.

For the purposes of this assignment, the similarity between two trees `tree1` and `tree2` is given by the highest similarity between a leaf in `tree1` and a leaf in `tree2`. In other words:

- Consider all possible pairs of leaves `(leaf1, leaf2)` where `leaf1 ∈ tree1` and `leaf2 ∈ tree2`.
- Each leaf node considered in the previous step is an organism (because the leaf nodes of the phylogenetic tree are individual organisms). Each organism has a genome sequence. Compute the similarity value between each pair of leaf nodes mentioned in the previous step as the similarity between the genome sequences of the corresponding organisms.
- From the set of similarity values computed in the previous step, pick the largest value. This is the similarity between the trees `tree1` and `tree2`.

Algorithm A3. Constructing a phylogenetic tree

There are many different algorithms for constructing phylogenetic trees. For this assignment, you should use the algorithm described below.

1. *Initialization.* Construct a list of trees `tree_list`, each of which is a leaf node containing a single organism.
2. *Iteration.* Repeatedly perform the following steps until `tree_list` contains just a single tree:
    a. Find trees `t1` and `t2` in `tree_list` (`t1 ≠ t2`) that have the highest similarity among all pairs of trees in `tree_list`.
    b. Create a new tree `t0` that has `t1` and `t2` as its two children.
    Important: To facilitate grading, set the left and right child of the new tree `t0` as follows:
        - If **`str(`**`t1`**`)`** `<` **`str(`**`t2`**`)`** choose `t1` to be the left child and `t2` to be the right child of `t0`.
        - Otherwise, choose `t2` to be the left child and `t1` to be the right child of `t0`.
    c. Remove `t1` and `t2` from `tree_list`.
    d. Add `t0` to `tree_list`.

    At this point, at the end of one iteration of the loop, the length of `tree_list` has decreased by 1.

Programming Requirements

1. You need a way to map the ID of an organism to its genome sequence as well as the N-grams obtained from that sequence. For this, implement a class **`GenomeData`** representing genome data for an organism. This should have at least the attributes listed below, together with the appropriate methods for accessing them:
    - **`_name`** or **`_id`**: the ID of the organism;
    - **`_sequence`**: the genome sequence for the organism; and
    - **`_ngrams`**: the set of N-grams for the organism's genome sequence.

    Put the code implementing this class in a file **`genome.py`**. You will have to import this code into your program using something like

```
from genome import *
```

2. Represent the nodes in the phylogenetic tree using a class **`Tree`**. This should have at least the attributes listed below, together with the appropriate methods for accessing them:
   - **`_name`** or **`_id`**: the name of the node. For leaf nodes, this should be the ID of the corresponding organism. For non-leaf nodes, one simple way to indicate that the node is not a leaf is to set this attribute to `None`.
   - **`_left`** and **`_right`**: references to the left and right subtrees of the node.

   Additionally, you may want to have an attribute at each node giving the set of IDs of all of the leaf nodes "under" that node: this can be useful when computing the similarity between two trees. This is not a requirement, just a suggestion.

   Put the code implementing this class in a file **`tree.py`**. You will have to import this code into your program using something like

   ```
   from tree import *
   ```

3. Use a list comprehension to compute the list of N-grams for each genome sequence (make sure that the last few elements of this list are not shorter than the value of `N` specified). (You can obtain a set of N-grams by applying **`set()`** to this list.)

4. Use a dictionary to map pairs of organism IDs to the similarity between those organisms.

5. To facilitate grading, set the children of each non-leaf node in the phylogenetic tree as as follows. During the iterative construction of the tree, when you combine two trees `t1` and `t2` as subtrees of a new tree node, if **`str(`**`t1`**`) < str(`**`t2`**`)`** then `t1` should be the left child; otherwise `t2` should be the left child of the new node.

Errors

There is no error checking required for this assignment.

Examples

Some examples are shown here.

Testing

This is probably the most complex program assigned in this class this semester, which makes it essential that you test your program thoroughly. I strongly urge you to use the testing strategies we have discussed in class. In particular, you should combine black-box testing, unit (white box) testing, and judicious use of **`assert`**s. Test your program as you develop it: because of the program's size and complexity, it will be harder to track down early-stage bugs if you wait all the way until the end of coding before you start testing.

For black box testing, you may want to start with small files created specifically for testing. For example, use several small files like tiny02.fasta: these files are small enough that you can manually compute N-grams and similarity values and check that the program is doing what is expected. Move on to larger input files only after working with several such small input files.

Files to be submitted

This problem requires you to submit three files:

1. `phylo.py`
2. `genome.py`
3. `tree.py`
