Biological Computation 20.181

Homework 4

Recall the basic formula for inferring gene trees:

```
for each possible tree:
    calculate score(tree)

report best tree
```

In this assignment, you will write Python code to evaluate the parsimony score of a particular tree. You will be provided with both the tree topology and the sequences of leaf nodes. We will need to expand our simple data structure to accomodate this new information associated with each node. For example,

```
tree = {'name':'a', 'left':None, 'right':None, 'data':chars }
```

where chars is a list of characters present at each position in our sequence alignment, for example...

```
chars = [ ['A'], ['G'], ['G'], ['A'], ['T'] ]
```

Why a 'list of lists'? This seems unnecessary for leaf nodes, where we know the exact sequence of nucleotides. But for internal nodes, there may be some positions, in which two or more nucleotides are equally consistent with the observed data. Such situations should be familiar from hw1. The 'initTree(tree,alignment)' function takes care of adding these data to our leaf nodes. You will write a function called 'downPass(tree)' which adds sequence data to internal nodes **and** keeps track of how many mutations (mismatches) are necessary to explain the oserved data.

The DownPass algorithm. The logic behind the 'downpass' is simple: if two daughter nodes share the same nucleotide at a given position, then the last common ancestor (LCA) also probably has the same nucleotide. Alternatively, if the two daughter nodes have different characters, then the LCA probably shares a common character with one (but not both) of the daughters.

If the two daughter nodes are also internal nodes, then each may have more than one possible character at each position of the alignment. In this case we can generalize the above logic using 'intersection' and 'union' operators. The pseudocode for this general algorithm is given below:

```
def downPass(tree):
#this function returns the number of mutations necessary
#to explain the sequence data, given the tree topology
#and assigns sequences to the internal nodes

    if tree is a leaf:
    #no need to assign sequence, its already there
       return 0 #no mutations necessary

    leftCost = downPass(left child)
    rightCost = downPass(right child)

    foreach character in sequence:
        if intersection(left child, right child) not empty:
```

```
            tree['data'] += intersection(left child, right child)
        else:
            tree['data'] = union(left child, right child)
            mutations += 1 #left and right trees disagree, so there must be at
least one mutation

    return mutations + leftCost + rightCost
```

Implement the downPass algorithm in the codebase provided.

The downPass algorithm successfully accounts for the minimum number of mutations necessary to explain a sequence alignment given a topology, yet the sequences at the internal nodes are not necessarily the best under the parsimony criterion. Why is this? (hint: the root node **is** optimal under parsimony -- what is different about the root and other internal nodes?)