Nathan Newbury **Writeup Hw6**

Brief algorithm outline:

Generating the root sequence:

```
    for i in range (k):
        choice = np.random.choice(np.arange(0, 4), p=gtr_probs)
        if choice == 0:
            seq = seq + 'A'
        elif choice == 1:
            seq = seq + 'C'
        elif choice == 2:
            seq = seq + 'G'
        elif choice == 3:
            seq = seq + 'T'
```

To generate the root sequence, we just draw a realization from our categorical distribution of πs which are stored in gtr_probs.

Evolve function main recursion:

```
for node in  tree.traverse_preorder():
        # only one parent for multi etc
        if node is tree.root:
            tot[node] = root_seq
        else:
            t = node.get_edge_length()

            parentseq = tot[node.get_parent()]
            P = sp.linalg.expm(t*Rnorm)
            seq = ""
            for letter in parentseq:
                #import pdb; pdb.set_trace()
                if letter == 'A':
                    row =0
                elif letter == 'G':
                    row = 1
                elif letter == 'C':
                    row =2
                elif letter == 'T':
                    row = 3
                #import pdb; pdb.set_trace()
                choice = np.random.choice(np.arange(0, 4), p= P[row])
                if choice == 0:
                    seq = seq + 'A'
                elif choice == 1:
                    seq = seq + 'C'
                elif choice == 2:
                    seq = seq + 'G'
                elif choice == 3:
                    seq = seq + 'T'
            tot[node] = seq
```

We traverse the tree in pre_order. This means we see parents before we see children. We find the length incident to the node and use this length to calculate the transition probabilities with $P = e^{tR}$ where R is our normalized transition matrix. We loop through each letter in the sequence saved at the parent node. For each letter, we look up the row that corresponds to that letter in our P matrix and draw from the distribution at that row. We then save the sequence at that node in a dictionary for future child nodes. At the leaves are our final evolved sequences.

Normalization of GTR transition rates:

It is important that the rate matrix is in the same units as edge length. It is convention to normalize the rate matrix to "mutation units". In mutation units, an edge length of 1 results in the expectation of 1 mutation. This means that d, the expected rate of mutations, should equal 1.

$d = \sum_i (-\pi_i R_{ii})$. We calculate d for our un-normalized matrix, then divide every element in our un-normalized matrix by d (R/d) such that $\sum_i (-\pi_i (R_{ii}/d)) = 1$. (R/d) is our normalized rate matrix.

```
    d = 0
    for i in range(len(gtr_probs)):
         d = d - gtr_probs[i] * R[i][i]
    row = 0;
    Rnorm = R/abs(d)
```

Time Complexity:

If we assume constant time for dictionary lookup and storage, constant time to draw from a distribution, and constant time to calculate the probability matrix, then our running time is only a function of the number of nodes and the length of our sequences. This means that the number of categories (A,C,G,T) never changes and dictionary lookup is optimized. In this case, for each node, we generate a sequence from the parent sequence by looping through each letter of the parent sequence. This results in a time complexity of **O(nk)**, where n is the number of nodes and k is the length of the sequence. It is important to note that for each node, we find a new probability matrix, which is a matrix exponential. Calculating the matrix exponential has a time complexity of $O(l^3)$(eigen-decomposition and matrix inverse are $O(l^3)$). If the R matrix increases in size, it would be the dominant factor over the sequence length. Running time is **max(**$O(nl^3)$**,** $O(nk)$**)**, depending on the sequence length and size of the rate matrix.