**Purpose:**

The purpose of the parser is to parse through variations of newick notation and create a distance matrix based on the taxa given via the unrooted star method. The parser was written in R.

**Methodology:**

**Setup phase:**

This parser is limited to binary trees in Newick notation but does not enforce a requirement of a binary tree for the root.

When first given the associated Newick notation, it will first parse and remove all numeric data until it is in its simplest form. In order to remove numeric data, we iterate through the data twice first removing branch length, and any unnecessary data then finally we will remove all extraneous whitespace. An example of this would be given the Newick form ((AN3C:.1,B1ED:.2)123:.15,(CAS2:.1,DE4D:.05 )125:.2) it will be reduced to ((AN3C,B1ED),(CAS2,DE4D)).

Upon successfully simplifying the Newick notation we will extract all the taxa names by subtracting the parentheses and splitting via commas and store them as a 1 by X character array where X is the number of taxa. The variable used to store these taxa are myVars and will be referenced multiple times through the program.

Upon finding the taxa we will preallocate a list of matrices called matrixList with length of myVars. It will then iterate via a for each loop through myVars and fill the matrixList with each index having three parts, a variable name(var), its priority(value), and a 1 by X matrix with X being the length of myVars(varMatrix). Priority will be used what variables to resolve first in the matrixList.

After initialization of the matrixList the program will then condense the Newick notation into whitespaces for the sake of identifying priority. For example ((A,B),C) becomes ((“ “,” “),” “) this will allow us to ignore the taxa in order to establish priority. We will then iterate through this character by character with a priority set to zero at the beginning and a variable called count set to one, if the char is a left parenthesis we increment the priority by one, if the char is a right parenthesis we decrease the priority by one. Finally, if the char is a whitespace we will store priority in the appropriate count index in matrixList and increment count by one. Note, as myVars/matrixList was initialized from left to right it is safe to assume that the first whitespace on the left corresponds to the first matrixList index, the second whitespace corresponds to the second matrixList index, and a similar pattern for the remaining whitespaces. The end result is a matrixList populated with their associated priority.

The final distance matrix is initialized at this point with it being X by X with X being the length of myVars. We will then set the column names and row names to the associated values in myVars.

**Algorithm**

Setting priority is important as we wish to resolve the innermost taxa first such as ((A,B),C). We would need to resolve A and B in a distance matrix before finally resolving it with taxa C. The program is set to be in a while loop to continue while the length of the matrixList is greater than two. In the while loop we will sort the variables by its priority and then make note of the highest priority variables in an indexList, that is preallocated with a default size of 100. There also exists a conditional if statement that will break out of the while loop if there are only three remaining matrixList values with the same priority.

Within the while loop after properly indexing the highest priority of matrixList we enter a for loop that resolves two indexes of matrixList at a time. Within the algorithm there exists two classifications of indexes of matrixList, an index with a distance matrix and an index with an empty matrix. This leads to three cases after looking at both variables, if there are only two taxas then the finalMatrix will enter the values of having a distance two apart, it will then delete both entries and replace it with a temporary variable, and populate a distance matrix with a distance of one from each of the two original taxa. The R code is setup to generate any temporary variable to start with tempvar0, temvar1, etc… The value of this is then reduced by one thus lowering its priority.

In the case of one taxa and one matrix it will iterate through the matrix and for each value greater than zero it will enter it to the corresponding finalMatrix with the taxa giving it the value in the matrix plus additional two for the total branch length. Afterwards a temporary variable will be created with the current non-zero values in the matrix being incremented by one with the taxa being added afterwards with a value of one.

In the case of two matrixes we will utilize a nested for loop, for each non-zero value in the first matrix it will iterate the second matrix and find non-zero values. Once the second non-zero value is acquired the appropriate slot in the finalMatrix will be filled with the first value plus the second value plus an additional two for branch length. Afterwards we will develop a temporary variable by combining the two matrix values and incrementing each value greater than zero by one.

Similar cases proceed for the last two variables however branch lengths are treated differently. Follow these above cases but instead entries into the finalMatrix are one less. The other case is with three final taxa/matrix groupings.

Within the four cases if all are taxa we will set a distance of two between all variables and enter it in the distance matrix. In the case of one matrix in two remaining taxa we will identify the index of the matrix and then enter it with the other two taxa for non-zero values plus two into the finalMatrix. Finally the last two taxa will be entered with a length of two apart. For two matrixes one taxa we use a similar example to how two matrix’s normally resolve and create a temporary matrix to then enter the finalMatrix values with the taxa. For three matrix it follows a similar step for two matrix but resolves the temporary matrix with the final matrix.