Foster Clark

Patrick Hough

Gary Sidoti

Abe Arce

**Computer Algorithms Project One**

The goal of this assignment is to input a text file containing a hierarchy of objects, construct a graph, and efficiently traverse this graph. In order to parse this file into the graph, we first call our import data function. This function reads in a single line at a time. Then this line is broken down by reading the first string, all the way up to the colon. We transfer this to all lowercase for uniformity, then this string is inserted as a vertex. Then in a while loop, we analyze the rest of the line, and break the string down at every comma. We then erase the space that is present immediately after the comma. Afterwards we then convert these strings to lowercase. For each string that is entered, we check if it is already a vertex. If the string is not a vertex yet, one is created for it. If it is already a vertex, it is a repeat and does not need to be created. For each vertex that is created, an edge is made from the source to all of its children. This will result in the graph being created as a hierarchy. Each vertex will have edges to its children, and those children are vertexes themselves, with children of their own. The graph stores this information using adjacency lists, so that it can be traversed easily. An adjacency matrix would be a waste of space, seeing as the graph is sparse. This makes it possible for us to traverse the graph with quickly and with ease.

The first problem asks us to traverse from a source vertex, down to a certain level, and return a number of subtypes the vertex has. This is done using a modified breadth first search. Instead of only inputting a source vertex, it also inputs the desired level of traversal, and the number of subspecies that the user wants returned. The idea of this function is to use a standard BFS, while keeping track of what level the current node is on. Unlike DFS, once one level is reached, all of the vertices that are on that level will be searched next. So if the user wants to go down to level two and return the first three subspecies, the BFS would traverse all of the species on level one, check to see what level it is on each step of the way, and only move to the next level if it is needed. Once the algorithm arrives on the desired level, it returns each vertex on the level, until the desired count is reached.

The second asks what is the most diverse sub species of a species. This is also found using a breadth first search modification. However in this one, each time a vertex is reached, the number of children is counted and stored alongside the name of the vertex. Each time a new vertex is reached, the number of children is compared to the current max. If the current node has more children, the current max is updated. At the end of the search, the vertex that has the most children is given as the output.

The third objective is to find the lowest common ancestor. Contrary to the last two algorithms, this takes two separate vertices as inputs. Instead of searching down like a breadth first search, we will work backwards from each vertex. As each new vertex is reached, it is stored, and all of the current storages are compared. If there are any matches, the lowest common ancestor has been reached.