Justin Moy

I used the Adjacency matrix as my data structure. It seemed the easiest to implement to me. Since the adjacency matrix has V rows and V columns with V being the number of vertices, it runs in O(V^2). In more detail, my program will check each vertex, it will then go throw every column in that vertex's row to see which vertices are adjacent to it. So it checks V columns for each row V times which is V^2 checks. Because of the implementation, the average and worse cases are both O(V^2). If every vertex is connected and needs to have the distance and parent updated each time, then it makes two updates that happen in the current time. For each vertex, all vertices are checked and updated if need be except the current one, which is 2(V-1) updates for all V vertices, so the worst case is O(2V(V-1)) = O(V^2). The average case is each vertex has a moderate number of edges but regardless, it checks V vertices for all V vertices which is still O(V^2). Using adjacency list would improve the average case because it will only check edges each vertex has so each edge gets checked twice because you still need to go down the list to find non-visited vertices and check if they need to be updated and each vertex is visited once making the average case O(V+E). The worse case is still O(V^2) because you will just end up with a V*V matrix except it's linked lists, there is no way to improve the worst-case because you must check all adjacent vertices to see if they have been visited and/or need to be updated. Worse case and the average case for memory are O(V^2) because the adjacency matrix will always create a V*V matrix, I did create another V*V matrix to hold edge values but that just makes it O($2V^2$) = O(V^2). Adjacency lists would improve the average case because it only takes memory based on the number of edges each vertex has making it more dynamic. So on average, each vertex won't be connected to all the other vertices in the graph. The worst-case stays the same for the adjacency list and I can't imagine a better way to do it, because you have to reserve memory for each vertex and its adjacent vertices which ends up being a V*V matrix-like structure.