Algorithms and Data Structures
Practicum 1 - Raspberry Pi Supercluster

Elianne Heuer, s4320514
Jordy Aaldering, s1004292

November 2019

Algorithm

First we read the number of vertices and edges from the input file using Stdin. Then we check whether one of the base cases can be applied: a vertex count of less or equal to 2, or an edge count less or equal to 1. If one of these base cases holds, 0 or 1 is returned respectively. Otherwise a graph is created in the form of an adjacency list. Next, the roots of the connected components are calculated by starting BFS at an arbitrary node to get a starting point. All visited nodes are marked as visited, so that each connected component is only checked once. BFS is then applied again from this starting point to get the longest path within that component. The node at the center of this path is selected and added to the list of roots, along with the distance of its longest simple path. Then the components are connected to each other by connecting all roots to the root with the biggest depth. Finally BFS is applied from an arbitrary node to get a starting point, and is then applied again starting at that point to get the longest simple path. That distance is then returned.

Correctness

We will start by taking a look at the two base cases. The first base case holds because if there are only two vertices, there cannot exist a path through another vertex and thus the distance will always be 0. If there is only one edge (and more than two vertices because of the first base case), then we know that we can connect all other vertices to one of these two vertices which already have an edge. This vertex connects to all other vertices, thus the longest path goes only through this vertex, causing its distance to be 1.

Now we will take a look at the actual steps of the algorithm. We know that we can apply BFS to find the longest path. We know that the longest path in a tree is always between two leaf nodes, because if there were a longest path from a non-leaf node, then that path could be extended to one of its leaf nodes. So if we now apply BFS from an arbitrary node, then we get the leaf node that is the furthest away. We know that this is an end point of the longest path, because if we were to extend the path one further away from the starting point, then this path would be the same, with one vertex added. Now that we have one end point we can apply BFS again starting in this point to get the other end point. We find the root by getting the vertex at the center of this path.

Connecting the roots also works. We know that the root with the biggest depth is always part of the longest simple path, because we have 3 cases: the longest simple path is in that tree, the longest path goes from a leaf node in this tree to another tree, or the longest path is between two other trees, passing through this root. From these cases it follows that connecting all roots to the root with the biggest depth gives the optimal construction.

Finally we apply BFS again just like before to get the longest path. We already know that this works, thus the algorithm indeed returns the distance of the longest simple path.

Complexity

Reading the input and creating the graph loops E times to add all of the edges, thus the time complexity is $\mathcal{O}(E)$. We also already know that the time complexity of BFS is $\mathcal{O}(V + E)$, which we will use later.

Computing the complexity of calculating the roots is more difficult, because BFS and the loop that it executes in share the same list of visited vertices. We can split this part in two cases, either we have many small connected components, or we have a few large connected components. In the first case the loop will loop V times, but because the components are so small BFS will be done in constant time. Thus in this case the time complexity will be $\mathcal{O}(V)$. In the second case the loop will only loop a few times, because BFS will visit most vertices, in this case BFS will dominate the complexity, causing it to be $\mathcal{O}(V + E)$. Thus in the worst case scenario the time complexity of this part will be $\mathcal{O}(V + E)$.[1]

Because it is possible that no nodes are connected, when connecting the components we could have V roots which we have to connect, causing this time complexity to be $\mathcal{O}(V)$. Finally, calculating the longest path is done by executing BFS twice again, giving it a time complexity of $\mathcal{O}(2(V + E)) = \mathcal{O}(V + E)$.

Then, combining all this we get that the time complexity will be $\mathcal{O}(E+(V+E)+(V+E))$. Which can be simplified to a final time complexity of $\mathcal{O}(V + E)$.

[1] We could also have a combination of these cases, but we do not know how we could calculate that.