Martin Ivanov Assignment 3f Graph Properties

1. **Question 1**

   To get the **eccentricity** of a vertex we run BFS on the graph with a start vertex **v** and after that for every single vertex we get the max distance which is done by calling the *distTo(**x**)* method of BFS where **x** is another vertex (if a new max distance is found we override the old one). The whole idea is to get the length of the shortest path from the start vertex **v** to the furthest vertex **x**. BFS takes time proportional to **E + V** and this will find the eccentricity for a single vertex. So if we want to find the eccentricity of all the vertices in a graph the time will be **O(V(V+E))** because we need to run BFS for each of the vertices.

   The **diameter** requires knowing the eccentricities. So we just run through all the vertices, we get their eccentricity and we find the largest eccentricity which will be the diameter.

   The **radius** is the opposite of the diameter where instead of finding the largest eccentricity we find the smallest.

   To find the **center** we need to know the radius, then we go through the vertices in the Graph and if the eccentricity of a vertex is the same as the radius then we know we have found the center.

   This being said in the constructor of **GraphProperties.java** we will run through all the vertices and calculate the diameter, radius, center and then assign their values to a corresponding instance variables. These values can be then returned in **constant time** via a corresponding getter method for each of them. For the eccentricity of the vertices I am using a hashmap mapping a vertex to a corresponding eccentricity. There is a method that takes as a parameter an **int vertex** which will return the eccentricity of this particular vertex from the hashmap in **constant time**.

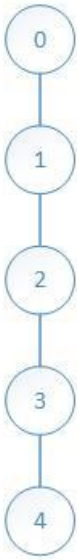
   This means that when an instance of the GraphProperties.java is made the preprocessing will do the job of calculating and after that from the client program we can execute the queries in **constant time.**

   Example: Graph (connected) with **9** vertices and **8** edges and the following connections: **0 1, 1 2, 2 3, 3 4, 4 5, 5 6, 6 7, 7 8** will return a **diameter of 8**, a **radius of 4** and a center being vertex **4** (we start calculating them from 0 so in normal terms this will mean vertex 5)

2. **Question 2**

   The solution identifies bridges which decomposes a directed graph into two-edge connected components. The time complexity is **O(E + V)**. The implementation uses the DFS algorithm to visit all vertices, identifying the “furthest” bridge first.

   Example: Graph with **5** vertices and **4** edges, and the following connections: **0 1, 1 2, 2 3, 3 4** will look something like that:



We can clearly see that each edge is a bridge as if we remove the edge pointing from 2 to 3 we will end up with having 2 disjoint subraphs. The same applies to all the other edges so in total this graph has 4 bridges. What I meant by identifying the “furthest” bridge first is that the bridge from 3 to 4 will be discovered first due to the **pre order** DFS which means that **v** vertices will be examined in pre order. The main points are the **lowOrder[]** array which is the minimum DFS preorder number of **v** and the set of vertices **w** for which there is a back edge **(x, w)** with **x** being a descendant of **v** and **w** being an ancestor of **v**.