AP Computer Science — Project 35d - Subgraphs
Dr. Paul L. Bailey — Monday, April 20, 2020

In this project, we work on subgraphs. I will add new methods to this project over the course of this week. Since these methods operate on more than on graph at a time, let us make them static methods in the `Graph` class.

Program 1. Create a method `public static Graph component(Graph G, Vertex v)`, which returns a graph which contains the associates in G of v, and all of the edges involving these vertices.

If v is not in G, this methods returns the empty graph. If G is connected, this method will return a copy of G, but if G is not connected, this method returns the largest connected subgraph of G which contains v.

Program 2. Create a method `public static Graph merge(Graph G, Graph H)` which returns a graph which contains all of the vertices and edges which are in G or H.

Program 3. Modify the `Vertex` class to implement the `Comparable<Vertex>` interface. One vertex is less than another if its id is less.

Program 4. Modify the `Edge` class to implement the `Comparable<Edge>` interface. Edge (a, b) are already adjusted so that you know $a < b$. Edge (a, b) is less than (c, d) if $a < c$, or if $a = c$ and $b < d$.

Program 5. Create a method `public static boolean areEqual(Graph G, Graph H)` which returns `true` if these graphs are equal, and returns `false` otherwise.

Note that two graph are equal if and only if they have their vertex sets are equal, and there edge sets are equal. It is important to realize that sets do not have multiplicity (no element occurs more than once in a set) or order (the order does not matter to a set).

So to do this, first check the the sizes of the vertex sets are the same. If they are, clone the vertices of the graphs, sort them, and then compare position by position. If that check passes, do the same with edges.