Visualizing Algorithms for Discrete Math education

We’ve all been there: the instructor in the discrete math course explains some lengthy proof, which may or may not make perfect sense inside his head, but certainly no sense is visible anywhere outside of it.

This project aims at doing something about that! The important fact, of course, is that many “proofs” in discrete math are really algorithms---the mathematicians like to call them “constructive proofs”, to confuse you ;).

Take, as a running example, the theorem that a graph is Eulerian (i.e., it has a closed Eulerian tour: a closed walk traversing each edge exactly once, and each vertex at least once) if it is connected and each vertex has even degree. There are multiple proofs for that theorem, but they are all really descriptions of algorithms producing a closed Eulerian tour.

In this software project, a computer program in Java will be developed, which is meant to be used in the classroom. It should allow to select a theorem and proof---but let’s start with one proof first, and let’s stick with the Eulerian tour theorem. The program would open a window displaying a graph. Then, it will proceed in constructing the closed Eulerian tour and visualizing it on the screen, but it will do so with user interaction: after each step, the student using the program needs to understand why the next step will be possible. For that, the logical arguments which force that must be visualized. In our example, a longer and longer tour is displayed on the screen. If there is, say, a vertex in the graph which is not (yet) in the tour, how can this be used to make the tour longer? It is in fact easy to visualize that argument answering that question on screen in such a way that the user has an active role in producing it! Students will never forget the proof once they have used that program.

There are numerous other proofs/algorithms which can, or rather, should be visualized in that way. We will stick with graph theory for starters. The software project has very easy ways to split work: Firstly, the visualization of the graphs requires some work which is somewhat independent from the algorithms themselves. Secondly, two sub-groups can work on two different algorithms.

The work will likely proceed something like this.

- You’ll start with that running example, seeing as you already know it. I’ll explain the logical arguments which need to be visualized, and my ideas how they should be visualized.
- You split into subgroups to implement the graph visualization, the algorithm itself, the additional computations needed for the logical arguments.
- You add the user interaction to it (the “active role in producing the logical argument).
- You produce the first prototype, test it, and make it look smooth, for classroom use.
• Take the next proof/algorithm—possibly split into 2 subgroups for 2 different algorithms.