This homework covers developing iterative algorithms. It is due in class Friday, April 11.

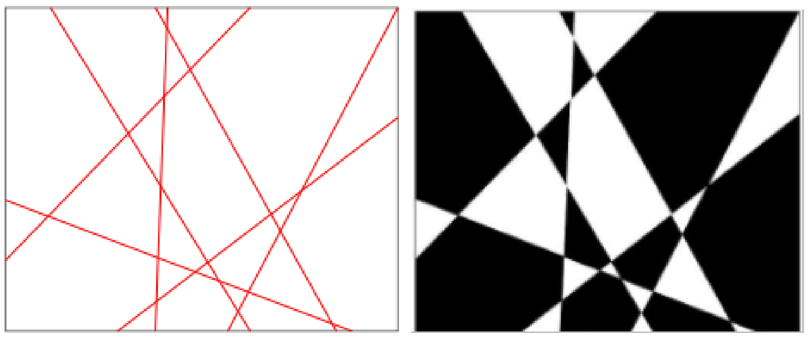
Write your solutions carefully — your work should be neat, readable, organized, and polished. As with writing a paper, you will likely need at least two drafts — the first draft can be rough (similar to what was written in class) as the focus is on going through the steps and finding a solution, while in the final draft the focus is on clarity of explanation and the quality of writing (similar to what was written after class). While it is not required, you are encouraged to type your work so that the revisions for the final draft are easier to make.

See the Policies page on the course website for information about revise-and-resubmit, late work, and academic integrity as it applies to homework.

For the first two problems, develop an iterative algorithm to solve the problem using the process discussed in class. Give each of the steps in the template — don't just give an algorithm. This is not a research task to look up the solution — the point here is to understand and be able to apply the process discussed in class to develop an algorithm yourself.

1. Given a subdivision of a plane defined by n lines, color each region either black or white so that any two regions sharing a boundary have different colors.

   An example is shown below — given the set of lines on the left, one possible coloring is shown on the right. (The other possible coloring is to swap white and black.)



2. Given a graph with maximum degree d, color it — that is, assign a color to each vertex so that no edge connects vertices with the same color — using at most $d+1$ colors.

The last problem is a little different style of problem — it's intended as a puzzle and the common iterative patterns identified don't seem to be applicable so our strategies for identifying avenues of attack don't really help. Nonetheless, the iterative algorithm development process can still help us escape the monster. Again, this is not a research task to look up a solution — the goal is to understand how you can leverage the iterative development process to help you figure out solutions to problems.

3. The setup:

   > You are in the middle of a lake of radius 1. You can swim at a speed of 1 and can run infinitely fast. There is a smart monster on the shore who can't go in the water but can run at a speed of 4. Your goal is to swim to shore, arriving at a spot where the monster is not, and then run away. How do you do this?

   To solve use the iterative development strategy to solve this problem, first view the swimming strategy through an iterative lens — the strategy is

   ```
   repeat
     swim a bit
   until the shore is reached
   ```

   Of course, "swim a bit" is too lacking in specifics for anyone to be able to escape the monster, but now we have a focus: figure out what "swim a bit" entails.

   Usually we go through the algorithm development steps in order, first defining the main steps and then establishing termination and correctness. But it is also possible to go the other way...

   (a) Given the starting point in the middle of the lake and the exit condition of reaching the shore, what would be a reasonable measure of progress?

   (b) A correct solution in this case escaping the monster, that is, arriving at the shore somewhere the monster is not. A wrinkle here is that there are many such places on the shore and it is much easier to deal with a more specific goal. What would in some sense be the best possible place to reach the shore? (Imagine you are very afraid of the monster — even if you can escape as long as you aren't where the monster is, where would you most rather be relative to the monster when you got to the shore?)

   (c) What loop invariant will ensure that arrival point? Hint: Keep in mind the template: the loop invariant combined with the exit condition should result in a statement that the result is correct. If the final statement should be "we've reached the shore at ..." and the exit condition is that we've reached the shore, the loop invariant needs to provide the ... part.

(d) The main steps need to make progress, but the loop invariant also has to be maintained. So, "swim a bit" can be broken down into two steps: "swim a bit to make progress" and then "swim a bit to make the invariant true again". State specific instructions for each of these "swim a bit" steps — what direction do you swim in and for how long?

(e) Solve the problem — give a swimming strategy for escaping the monster, in the form of an iterative algorithm. Write this up according to the iterative development template as you've done for the other exercises but you can omit the "identify avenues of attack" and "determining efficiency" steps as those are not really applicable here — the notion of running time doesn't make sense and our avenue of attack was to work backwards from what we need to establish about the main steps — termination and correctness — to find the solution rather than to exploit a known pattern.