This homework covers designing iterative and recursive algorithms. It also revisits modeling with graphs. #4 (ski-o) is due in class Friday, April 3. The rest of the problems are due in class Wednesday, April 8.

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

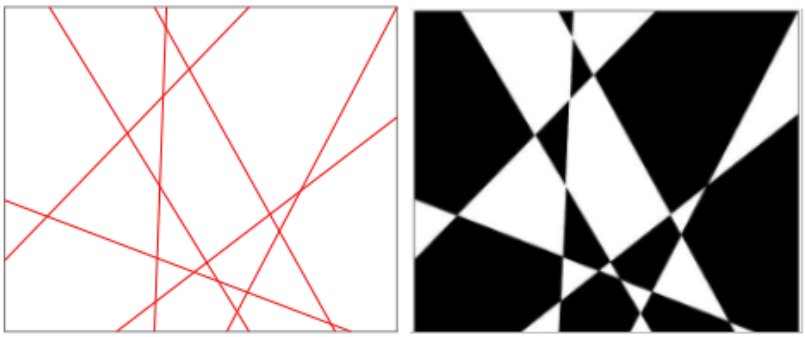
Write your solutions carefully — your work should be neat, readable, and organized. Keep in mind that what you hand in should be a presentation of your work, not your train-of-thought stratch work with a solution mixed in somewhere.

---

1. Do the homework #8 drill problems on Canvas. (Look for `hw8 drill` in the Quizzes section.)

2. Develop an iterative algorithm to solve the problem stated below using the process discussed in class. Show your work — and your understanding of the steps in the process — by including each of the steps in your writeup. Don't just give an algorithm! See the posted writeup for the assigning people to groups problem from class for an example. (Don't include the marked "commentary" parts.)

   > 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. Your algorithm only needs to produce one valid coloring, so either is fine.)

3. This problem illustrates both that the iterative algorithm development process discussed in class can be applicable in a variety of situations, and that it is not necessary to follow the steps in order.

> 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?

Paradigms and patterns. This can be viewed in a "produce output" light — the goal is a route to swim to escape the monster, and one can think of building of this route a bit at a time:

```
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.

Often we go through the algorithm development steps in order, first defining the main steps and then establishing termination and correctness. But this isn't required — steps can be tackled in any convenient order.

(a) *Measure of progress.* 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) The goal is to escape the monster, that is, to arrive 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) *Loop invariant.* What loop invariant seems like it help you reach that arrival point?

Hint: Keep in mind the "final answer" step of the correctness argument — a strategy for the loop invariant is to be about the correctness of the solution so far, so that when the exit condition reached, the solution so far is the whole solution and the loop invariant leads directly to the correctness of the final answer. If you want X to be true when the exit condition is reached, try making the loop invariant a statement that X is true for the partial solution.

(d) *Main steps.* 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" (according to the measure of progress)

and then "swim a bit more 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) *Establish and maintain the invariant.* Is it always possible to swim a bit more to make the invariant true again? Explain.

(f) *Exit condition.* Exit the loop when it is no longer possible to maintain the invariant. When is this, in terms of the measure of progress?

(g) *Wrapup.* If the loop exit isn't at the shore, how do you swim to get to the shore? (Avoiding the monster.)

(h) *Algorithm.* Put it all together — how do you escape the monster? (You should be able to give the strategy in three steps — combine all of the "swim a bit to make progress" steps into one, combine all of the "swim a bit more to make the invariant true again" steps to get to the point where escaped is assured, and then do the wrapup step to get to the shore and escape.

4. Give an algorithm for the ski-o problem described below by modeling it as a graph problem. Address the following things in your solution:

   (a) What do the vertices of the graph represent? The edges? What is the solution to the problem in terms of the graph?

   (b) Is the graph undirected or directed? Weighted or unweighted? (If weighted, what do the weights represent?) Simple or not simple? Sparse or dense? Cyclic or acyclic? Embedded or topological? Implicit or explicit? Labeled or unlabelled? Provide a brief explanation with each answer.

   (c) Give an efficient algorithm for solving the problem in terms of a well-known graph algorithm (i.e. one discussed in class). Design graphs, not algorithms! Imagine that you have a library of implementations of well-known graph algorithms that you can use — design your graphs so that you can use them as-is instead of having to modify an algorithm or implement your own.

Ski orienteering is the sport of cross-country navigation on skis — competitors must navigate to a series of checkpoints (called controls) using a map and compass. An orange-and-white flag marks the location in the terrain. Controls typically must be visited in a particular order, and the goal is to navigate to a certain sequence of controls as quickly as possible.

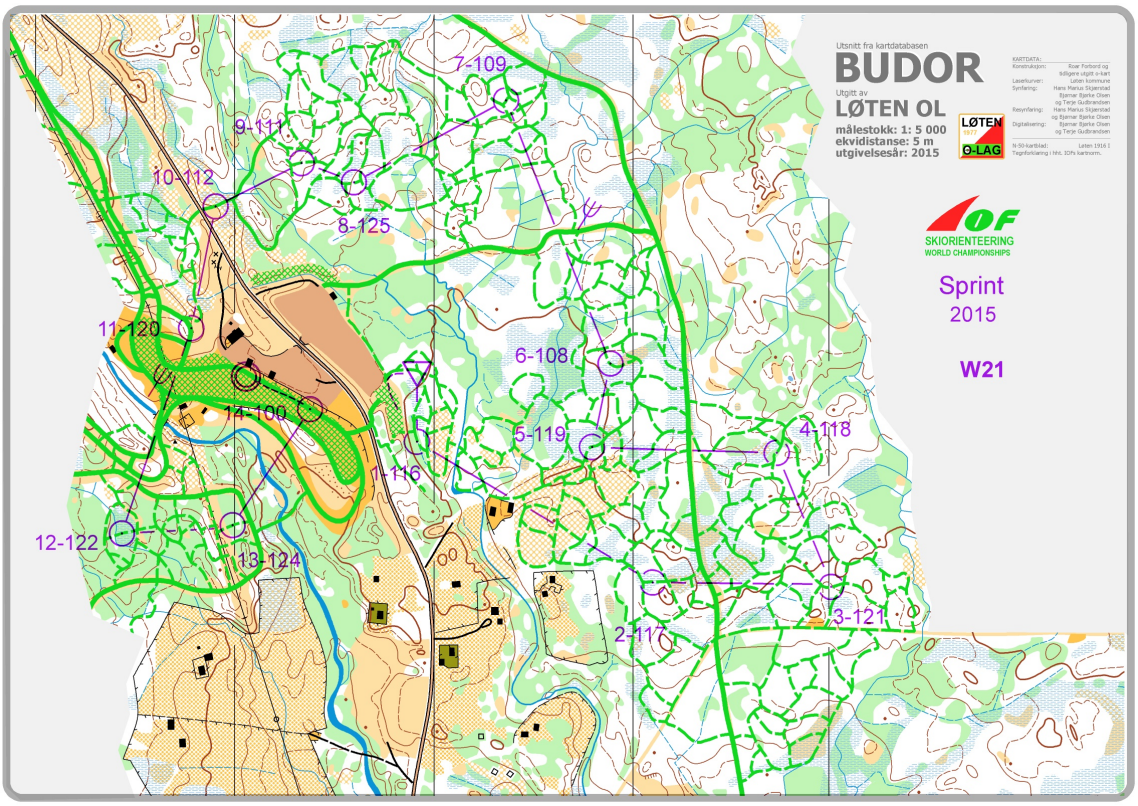
The task: Given a map which provides information about the track network (junctions, track segments between junctions, and the time it takes to ski each track segment) and a course consisting of a start, a finish, and a sequence of control locations, find the fastest route from the start to the finish that visits all of the controls in the specified order.

You can make the following assumptions for this problem:

- The start and finish are at track junctions.
- All other controls are along tracks rather than at track junctions. They can be treated as being in the middle of the track segment (halfway between junctions).
- It is possible to change directions at controls and return the way you came without continuing on to the next junction. It is not possible to change direction in the middle of any other track segment.
- No shortcuts — you must stay on a track and not cut between tracks.
- There aren't any mandatory routes that must be followed.
- There are no one-way trails, but note that the speed of travel in one direction may be very different from the other direction. (e.g. going in the uphill direction vs the downhill direction)

It is OK to visit a control more than once (or visit controls not on your course) as long as there is a subsequence of controls visited that contains the right controls in the right order — it is not necessary to avoid skiing by other controls.

If you are curious about the maps, read on...



Maps for ski-o show the track network, terrain, vegetation, and the course:

- Tracks are shown with green lines. The symbol indicates the size of the trail — dashed lines are the narrowest and typically slowest, thick solid lines are the widest and typically fastest.
- Contour lines are in brown.
- Vegetation is shown with shades of yellow and green.
- The course is in purple — a triangle for the start, circles for the controls, and a double circle for the finish. Straight lines between control circles and the first part of the number labelling each circle indicate the order in which the controls must be visited. (The second part of the number is the control code, which lets you verify that the flag you've found is the correct one.) Dashed purple lines indicate a marked (and mandatory) route.