Revisiting Step 1 | Coursera

Revisiting Step 1

Work an Example Yourself

The first step to devising an algorithm is to work an instance of the problem yourself. As we discussed earlier, if you cannot do the problem, you cannot hope to write an algorithm to do it—that is like trying to explain to someone how to do something which you yourself do not understand how to do. However, you have to not only be able to do the problem, but also do it methodically enough that you can analyze what you did and generalize it.

Often, a key part of working the problem yourself is drawing a picture of the problem and its solution. Drawing a clear and precise picture allows you to visualize the state of the problem as you manipulate it. Having a clear idea of the state of the problem, and how you are manipulating it will help you with the next step, in which you write down precisely what you did on this instance of the problem.

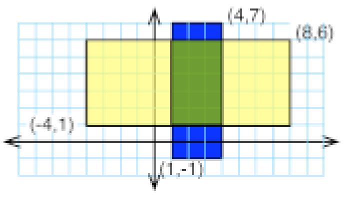
We will use the following problem as an example to work from for the rest of this chapter:

Given two rectangles, compute the rectangle which represents their intersection.

You may assume the rectangles are vertical or horizontal.

The first thing we should do here is to work *at least* one instance of this problem (we may want to work more). In order to do this, we need a bit of domain knowledge—what a rectangle is (a shape with 4 sides, such that adjacent sides are at right angles), and what their intersection is (the area that is within both of them).

What instance we pick is really up to us. For some problems, some instances will be more insightful than others, and some will expose *corner cases*—inputs where our algorithm needs to behave specially. The most important rule in picking a particular instance of the problem is to pick one that you can work completely and precisely by hand.



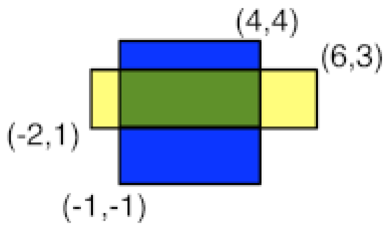
Working an example of the rectangle intersection problem

The figure above shows the results of Step 1 for the rectangle intersection problem. We picked an instance of the problem—here the yellow-shaded rectangle from (-4,1) to (8,6) intersecting with the blue-shaded rectangle from (1,-1) to (4,7). The resulting intersection is the green-shaded rectangle from (1,1) to (4,6).

You should note a few things about this example. First, while the yellow/blue/green coloring is not truly a part of the problem, there is nothing wrong with adding extra information to your diagram to help you understand what is going on. Second, note that the diagram is done precisely—we drew a Cartesian coordinate grid, and placed the rectangles at their correct coordinates. This precision not only ensured that any information we obtained from analyzing our diagram was correct and not a result of sloppy drawing (though whether some relationship is generally true, or a consequence of the specific case we chose is *not* guaranteed by a careful drawing). You typically do not need to draw things with draftsman-level precision, but the more precise you can be, the better.

In this case, we can tell the answer just by looking at the picture and seeing where the green region is. However, to write a program to do this, we need to figure out the math behind it—we need to be able to work the problem in some way other than just looking at the diagram and seeing the answer. Sometimes trying work things mathematically is hard when you can just see the answer. Learning to put the obvious aside and think about what is going on is a key programming skill to learn, but takes some time.

If you struggle with it, it may be useful to work another instance of the problem, but eliminate extra information that lets you jump straight to the answer without understanding. The figure below shows a different instance of the rectangle problem with the Cartesian grid removed (note that it was still drawn such that the rectangles are the right size and in the correct relative positions). We can still precisely work the problem from this diagram, but it is a little harder to just look at the Cartesian grid and see the answer. Take a second to work out the answer before you continue.



Another instance of the rectangle problem, with the Cartesian grid removed

Note that there is nothing wrong with working a few instances of the problem, taking different approaches as you do it, and including/excluding various extra information as you do so. In general, it is better to spend extra time in an earlier step of programming than getting stuck in a later step (if you do get stuck, you might want to go back to an earlier step and redo it with another instance of the problem). For Step 1, doing a few different instances of the problem is preferable to moving into Step 2 and only being able to come up with "I just did it—it was obvious."