What is an algorithm?

Principles of Computing (15-110)

Instructor: Giselle Reis

Lecture on Sunday 12th January, 2020

In our daily and professional life we are often faced with problems to solve. These can be as small as choosing a snack at the cafeteria, or as complicated as making the course schedule for the next semester. When trying to find solutions, we need to consider a number of different aspects of a problem, such as:

- **Constraints**

  For example: Maybe you only have 2 minutes to make a choice at the cafeteria before your next appointment, or you only have 5 riyals to spare. A solution is only valid if it satisfied the constraints.

- **Available knowledge**

  For example: You want to take the healthiest snack, but only a few of them have nutritional information. You need to find a solution with the information that you have at that moment.

- **Solution requirements**

  For example: The snack could be anything that fills you up, or the tastiest one. Do you want *a* solution, a *good* solution, or the *best* solution?

Once we understand the problem, its constraints, the information we can use, and what consists of a solution, we are in the position of *solving* it. Choosing a snack is an easy problem to solve, and we can do this in our heads. But other problems are too complex for us to handle, such as the scheduling one. Imagine there are hundreds of courses to be allocated. Each of them have a time frame (full semester, first half, or second half), a frequency (how many times a week), a duration (how long each session takes), and an instructor (or two). There are a limited number of classroom, and some courses may not overlap with each other because some students need to take both of them. Moreover, each instructor cannot hold two different classes at the same time. Some courses need to be in classrooms, others in labs. How would you go about solving this? There are potentially millions of different configuration of classes, but we need to find one that satisfies all the constraints.

Fortunately, *computers* can help us solve that! Computers are good at such kinds of problems, the ones that would take us forever, but that, if we had forever, we could eventually solve. In the example above, if I give you three courses with its constraints, you can probably figure out a schedule with some patience. If you can spell out, step-by-step, on how you found that schedule, and if this process can be generalized for any number of courses, we can give that to a computer. The computer will execute these steps ultra fast, and will give you a solution for three courses within the blink of an eye. But more importantly, it will be able to solve it for one hundred courses in less than one minute, which would take you or me days or months!

This course is precisely about learning how to go from a problem, find out how solutions can be found, spell this process as an algorithm, and program it so the computer can solve for you.

$$\text{problem} \longrightarrow \text{how to find solutions} \longrightarrow \text{algorithm} \longrightarrow \text{program}$$

Definition 1. *An algorithm is a sequence of instructions for computing an answer (output) for a problem (input).*

Remember that computers are just machines that will execute the instructions as given, no more no less. In the example above, you could have found a schedule for three courses just by trial and error, and then your sequence of instructions would be:

1. Place the courses on a schedule
2. Check if it is a solution
    (a) If yes: this is the answer
    (b) If no: go back to step 1

If we try to follow these instructions *ipsis litteris*, we may end up never finding a solution at all, since there is nothing preventing us from trying out the same configuration over and over again. Also, **computers do not know how to choose things at random**, so we need to be more specific about what "place course on a schedule" means. We leave as an exercise specifying these instructions in a more systematic way.

> Good algorithms compute the correct solutions, do so efficiently (no useless steps), and are easy to understand and modify.

Let's look at a simpler example: how to compute ages. You know today's date, and your friend tells you they were born on 16/04/1998. Can you calculate their age? Of course! The sequence of steps you have followed to do this might have been more or less:

1. Subtract the current year from the birthday year
2. Check if the current month is greater than the birthday month
    (a) If yes: the solution is the number computed on step 1
    (b) If no: check if the current month is equal to the birthday month
        i. If yes: check if the current day is greater than or equal to the birthday day
            A. If yes: the solution is the number computed on step 1
            B. If no: the solution is the number computed in step 1 minus 1
        ii. If no: the solution is the number computed on step 1 minus 1

Exercise: can you write a sequence of steps for a person to draw a square? What about a house? Or a boat?