Algorithms And Data Structures I Course Notes

Felipe Balbi

October 10, 2019

Week 1

1.1.1 What is a Problem? What is an algorithm?

In computing we deal with problems that are addressable by computers. In other words, we deal with problems that are **computable**. The underlying language used to communicate with a computer needs to be mathematical, regardless of which *Programming Language* we use.

Computers require each and every idea to be converted into a mathematical concept (a number or a truth value).

Toy example:

```
x days of holiday total
y days of holiday used
x > y ? True  or False
x - y = Amount of days left
```

Note that in the case of `x-y` a positive result implies *True* while a negative or zero result implies *False*.

Problems can usually be solved in more than one way. The study of Algorithms and Data Structures gives us tools to decide which method is *better* than the other.

Definition 0.1 (Algorithm) *A general and simple set of step-by-step instructions which, if followed, solve a particular problem.*

Keep in mind that it's highly desirable to have a general purpose algorithm that solves many instances of similar problems. For example, instead of having an algorithm to solve $x^2 = 2$, it would be better to produce an algorithm to solve $x^2 = y$ given x and y are in $\mathbb{Z}$.

1.2.1 Al-Khwarizmi and Euclid

Algorithms predate the digital computer by hundreds of years. The word *algorithm* comes from the latinized name of Persian polymath **Al-Khwarizmi** (written as *algorithmi*).

One of the first known algorithms is Euclid's algorithm for calculating the *Greatest Common Divisor* between two numbers. It was described around 300 B.C.

An algorithm is a **mathematical concept** that can be instantiated as a computer program.

1.2.4 From mathematics to digital computers

Before we think about how to concretely describe an algorithm, we need to consider how to write the input data into a computer. It is **not** always possible to input arbitrary data into a computer. For example the number π is an irrational number (actually, it's transcendental see [1], which means it has an infinite decimal expansion; therefore we can't input π into a computer, as computer memory is a finite resource. We can only approximate it.

When approximating irrational and transcendental numbers with rational numbers, we will be left with an error in our calculations. This error is referred to as the *precision* of our calculation. The smaller the error, the more precisely correct our computer handles the input to our problem.

The need for approximations came to be before digital computers. The Egyptian-Greek mathematician, Heron of Alexandria, produced an algorithm for calculating and approximation of the square root of a number. That algorithm is called Heron's Method.

Heron's Method

Say we want to calculate $x^2 = 2$, give x to 1 d.p.

We **know** x must be $1 < x < 2$, we take the mean to get a candidate:

$\frac{1+2}{2} = 1.5$

$x_g = 1.5 \rightarrow x_g^2 = \frac{9}{4} > 2$

The answer **must** be within the interval $1 < x < 1.5$.

Thus:

$$\begin{aligned} \frac{2}{x} = x < x_g \rightarrow \frac{2}{x_g} < x \\ \rightarrow 1.\dot{3} = \frac{4}{3} < x < \frac{3}{2} = 1.5 \end{aligned}$$

Take mean to get new candidate:

$x\prime_g = \frac{17}{12} = 1.41\dot{6}$

correct to 1 d.p.

We can repeat this process as many times as we want in order to increase accuracy.

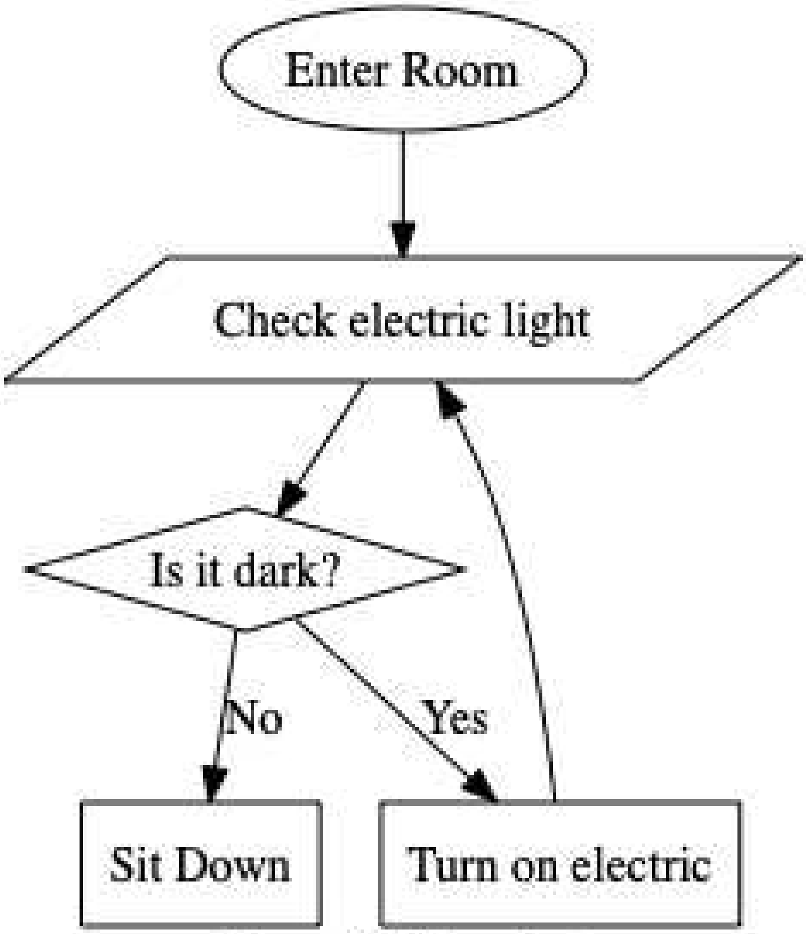
[1] https://www.youtube.com/watch?v=WyoH_vgiqXM

Week 2

1.3.1 Introduction to flowcharts

Using flowcharts to describe algorithms. Flowcharts are abstract representations of processes such as workflow and project management. They are composed of differently shaped boxes and arrows connecting them. Boxes typically represent actions, referred to as *activities*, *states of affairs* or *decisions*. Arrows represent workflow or outcomesthat result from the decisions.

Let's look at an example below:



Example Flowchart

Flowcharts use different shapes for different meaning:

- Oval

  Ovals are Terminal nodes. They are used either as *Start* or *End* of an algorithm.

- Parallelogram

  These represent I/O actions, like gathering or displaying data.

- Arrows

  Represent control flow of the algorithm by connecting one node to another.

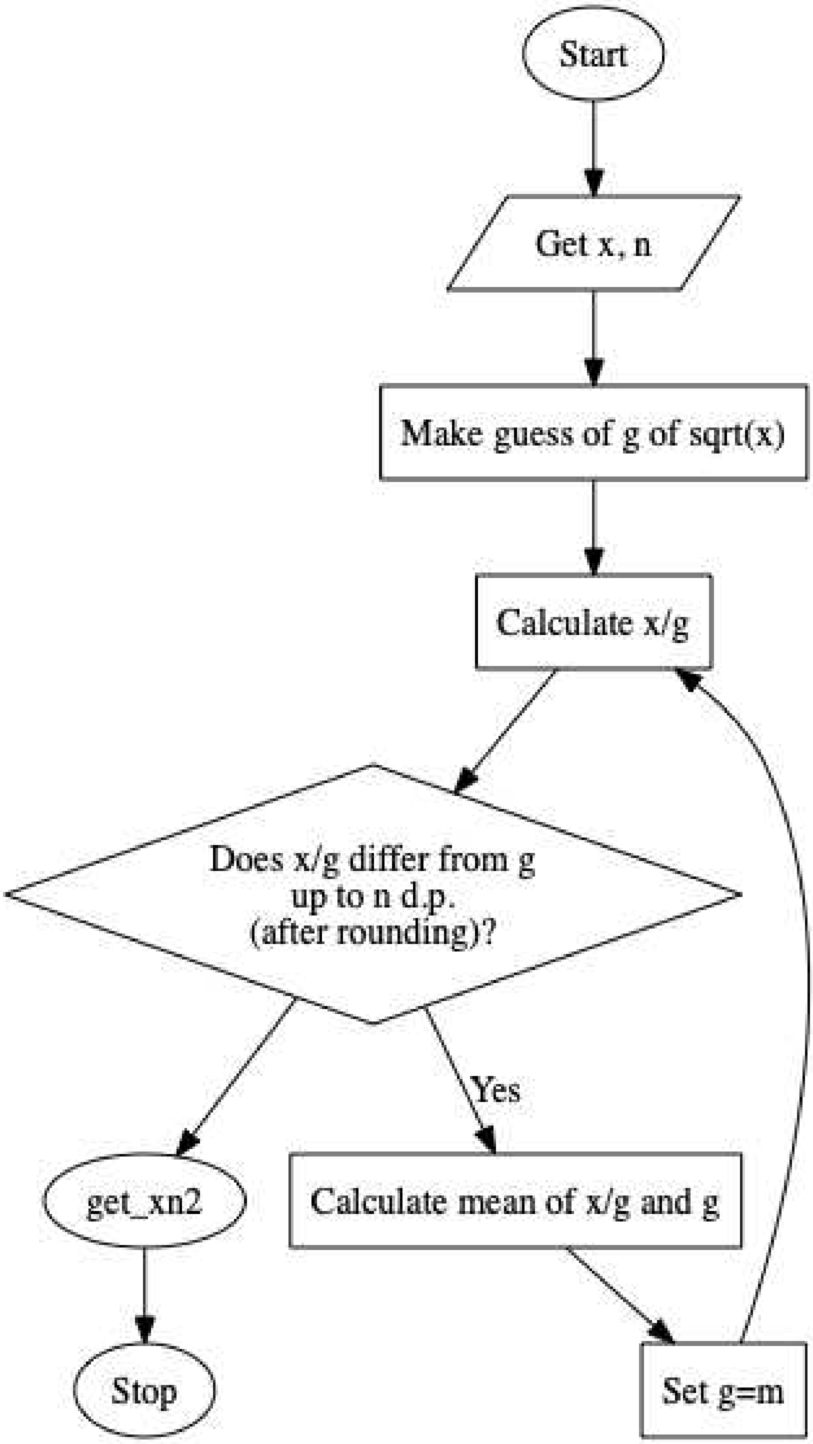
- Diamond

  Diamonds represent decisions blocks. Typically they have two outcomes: Yes/No, True/False, etc.

- Rectangle

  Basic actions, turning on the light, are carried out by rectangle boxes.

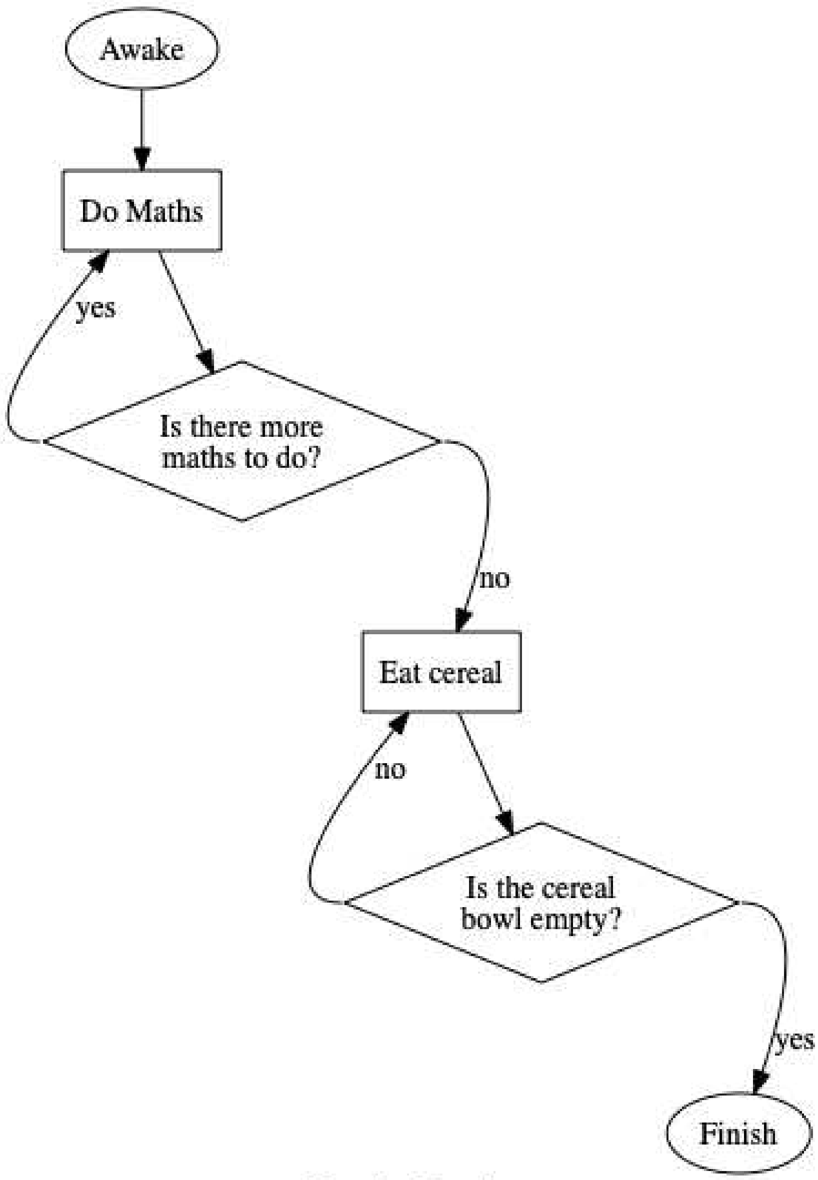
Heron's Method Flowchart



Heron's Method

1.3.4 My example of my morning routing as a flowchart

Professor Matty showed a flowchart of his morning routine. It went something like below:



Morning Routine

1.4.1 Conclusion

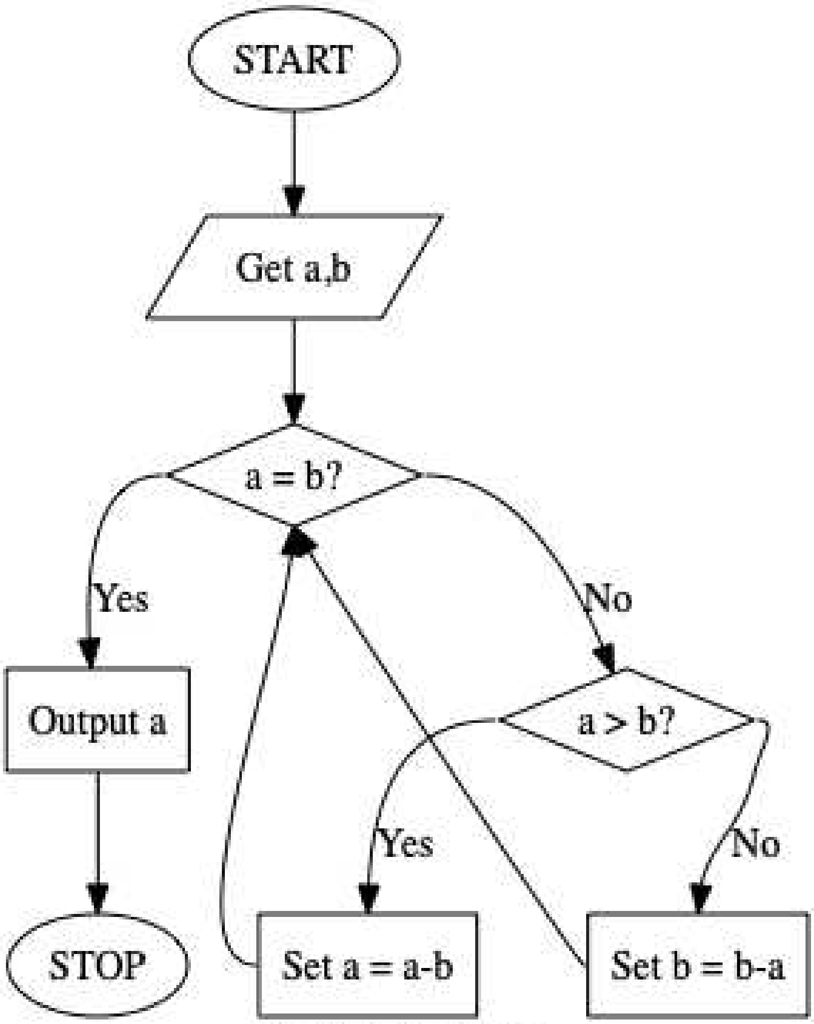
We learned concepts of problems, solutions, and algorithms. We learned a bit of algorithmic history and how to describe algorithms in flowcharts.

Week 3

Pseudocode is a different way of describing algorithms that resembles a programming language.

2.0.1 Solution to the Birthday Party Problem

The problem to be solved here is the problem of finding the *Greatest Common Divisor* of two numbers. Euclid's Algorithm is one possible algorithm for finding the GCD of two numbers.



Euclid's Algorithm

2.0.3 Introduction to Topic 2

Pseudocode is a simple way of describing and illustrating an algorithm in a language that resembles computer programs. It's widely used by Computer Scientists to discuss algorithms.

While flowcharts are an excellent way of conveying the flow of information in an algorithm and clearly showing the particular operations that are happening, drawing flowcharts is cumbersome and time-consuming.

Also, when we need to implement an algorithm, having a representation that resembles a general programming language is far better as it makes the process a little easier.

2.1.1 Discretisation and pseudocode

Discretization is the process of taking a continuous quantity and turning it into discrete steps. For example, if we were to build SW to control a thermostat until a desired temperature is reached, we would have to provide the program with discrete temperature steps (i.e. 0.5 °C) which would be incremented or decremented until the desired temperature is reached.

For this reason, pseudocode is a natural way to describe algorithms since they closely resemble computer programs. Pseudocode employs standard mathematical symbols along with a few extra bits of special notation. One such bit is the assignment symbol.

Algorithm 1 The assignment symbol

$x \leftarrow 2$

$y \leftarrow$ TRUE

When using pseudocode, we should refrain from naming variables with terse names such as x, y, z, etc. As algorithms get large, it becomes to track down which letter is used for what value. Instead, we should use descriptive names such as *DesiredTemperature* for the thermostat example above. The only constraint here is that we never add **spaces** to variables names.

We read pseudocode much like English, where the order goes from left to right and from top to bottom. Assignments are also *self-referential*, which means that after a variable has been assigned a value, we assign a new value based on the variable itself:

Algorithm 2 Self-referential

$x \leftarrow 2$

$x \leftarrow x + 3$

Common symbols used in pseudocode

- Assignment: $\leftarrow$

- Arithmetic operators:
  - Addition: $+$
  - Subtraction: $-$
  - Multiplication: $\times$
  - Division: $/$
- Comparison operators:
  - Equality: $=$
  - Difference: $\neq$
  - Less than: $<$
  - Greater than: $>$
  - Less than or equal to: $\leq$
  - Greater than or equal to: $\geq$
- Logical operators
  - And: $\wedge$
  - Or: $\vee$
  - Not: $\neg$
  - if ... then ... end if

Algorithm 3 if ... then ... end if

$x \leftarrow 2$
if $x > 1$ **then**
    $x \leftarrow x - 1$
end if
if $x < 0$ **then**
    $x \leftarrow x + 1$
else
    $x \leftarrow x + 10$
end if

Example: Thermostat pseudocode What follows is a simple example of a real pseudocode to implement a simple algorithm that increased the temperature of a thermostat by half a degree if it's less than a threshold.

Algorithm 4 Thermostat

$temperature \leftarrow 18$
$desired\_temperature \leftarrow 20$
if $temperature < desired\_temperature$ **then**
    $temperature \leftarrow temperature + 0.5$
end if

2.1.2 Pseudocode and functions

Pseudocode replicates other concepts from programming languages. One such concept is that of functions.

Function is a very general concept in computer science and mathematics. Functions take inputs and return outputs. For example the sum function takes two numbers as inputs and returns one number as output. Functions can return other types of values, such as Boolean values. A predicate is a function that returns a Boolean value given some input.

Here's an example function `EVEN`

Algorithm 5 The Even function

```
function EVEN(n)
    if n mod 2 = 0 then
        return TRUE
    else
        return FALSE
    end if
end function
```

Here, everything inside `function` and `end function` is referred to as the *body* of the function. The body is composed of *statements*. We have an `if-then` statement with a `return` statement inside of it.

A `return` statement is terminal, meaning `return` causes the function to stop.

2.2.1 Introduction to loops in pseudocode

Iteration is the idea of repeating something multiple times. Iteration is also to as *looping*. The two main looping structures are *for* loops and *while* loops. In *for* loops, we initialize a variable (e.g. i) to be our loop counter. At each iteration of the loop, we increment i by 1 until it reaches a target value, such as 10.

Algorithm 6 For Loop Example

```
x ← 1
for 2 ≤ i ≤ 10 do
    x ← x + i
end for
```

The text between **for** and **do** is referred to as the *condition* of the for loop. The text between **do** and **end for** is called the *body* of the for loop.

The basic concepts expressed in the context of the for loop, also apply to the while loop, just the structure is a little different. Here's the same algorithm from the for loop, implemented using a while loop:

With all this new vocabulary, we can implement the algorithm using pseudocode

Algorithm 7 While Loop Example

```
x ← 1
y ← 0
while x < 11 do
    y ← x + y
    x ← x + 1
end while
```

Algorithm 8 If $x^2 = n$ is x an Integer?

```
function IsXInteger(n)
    y ← FALSE
    for 1 ≤ i ≤ n do
        if i^2 = n then
            y ← TRUE
        end if
    end for
    return y
end function
```

The problem with this approach is that we may be squaring numbers well over than is necessary. A slight improvement can be achieved with a *while* loop.

Algorithm 9 If $x^2 = n$ is x an Integer? - While loop

```
function IsXInteger(n)
    y ← FALSE
    i ← 1
    while i^2 ≤ n do
        if i^2 = n then
            y ← TRUE
        end if
        i ← i + 1
    end while
    return y
end function
```

We could also employ *break* and *continue* statements to make this algorithm a little bit better. *break* stops a loop altogether while *continue* skips to the next iteration.

Algorithm 10 shows an example of both.

We should try to avoid *break* and *continue* in pseudocode.

2.2.2 Euclidean algorithm in pseudocode

With all this knowledge, we can write the Euclidean Algorithm in pseudocode. Please, refer to algorithm 11.

Algorithm 10 Break and Continue

```
x ← 1
y ← 10
while x < 11 do
    if y = 10 then
        x ← x + 1
        y ← y − 1
        continue
    end if
    break
end while
```

Algorithm 11 Euclidean Algorithm

```
function GCD(a , b)
    while a ≠ b do
        if a > b then
            a ← a − b
        else
            b ← b − a
        end if
    end while
    return a
end function
```