Algorithms

Author(s)
Silvio Peroni – silvio.peroni@unibo.it
Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

Abstract

These lecture notes introduce the notion of *algorithm* and *pseudocode*, so as to provide the initial tools for instructing a computer in executing a particular task. In addition, a particular kinds of graphical pseudocode is introduced, i.e. the flowchart. The historic hero introduced in these notes is Ada Lovelace, considered the first computer programmer. Her work in translating and commenting a scholarly paper describing Babbage's Analytical Engine has been one of the most important milestones of the Computer Science discipline.

Historic hero: Ada Lovelace

Ada Lovelace (shown in Figure 1) was a daughter of the poet Lord Byron, and was an English mathematicians who became famous for her work on the Babbage's Analytical Engine. Despite her father's habits, her mother, Anne Isabella Milbanke, strongly promoted Ada's interest in logic and mathematics, even after the death of her father. One of the goals of her mother was to prevent her to incur in the same insanity that characterised her father's life. However, the creativity that was intrinsically tied up on Byron family manifested in a totally unpredictable way.

In 1833, she attended a party organised by Charles Babbage for presenting its Difference Engine. She was so impressed by Babbage's invention that she started a correspondence with him that spanned 27 years [Morais, 2013]. She was the English translator of the first article about the Analytical Engine, that was written in French by Luigi Federico Menabrea, and that she enriched with several annotations. Among these annotations, there was a description of

how to use the Analytical Engine to calculate the Bernoulli numbers [Menabrea, 1842]. Technically speaking, this was the first computer program ever written (actually the first *algorithm* of the whole history) and it has been created by Ada without even having the real machine implemented – since the Analytical Engine was just a theoretical machine that was not physically built by Babbage.

Figure 1. Portrait of Ada Lovelace. Source:
https://en.wikipedia.org/wiki/File:Ada_Lovelace_portrait.jpg.

However, her vision about the possible uses of the Analytical Engine went even further [Morais, 2013]:

The operating mechanism can even be thrown into action independently of any object to operate upon (although of course no *result* could then be developed). Again, it might act upon other things besides *number*, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine. Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

That "science of operations" is a reference to a particular field that was clearly named and identified only after several years. In practice, Ada Lovelace was talking about Computer Science one hundred years before its formal introduction. For her work in the field, Ada Lovelace is recognised as the **first computer programmer in history**.

Algorithms and programmers

Before to introduce the main topic of these lecture notes, it would be worth to focus on simple examples we usually face during our daily life. Figure 2 illustrates two examples of step-by-step procedures we have to follow for preparing canapé crackers and for assembling a particular lamp respectively. While the actual goal of the two examples is extremely different, since the first one is a recipe while the other one is a set of instructions for assembling an utensil, they are described in terms of a shared abstract notion: instructions for *producing something* starting from some *initial material* we have, i.e. an algorithm.
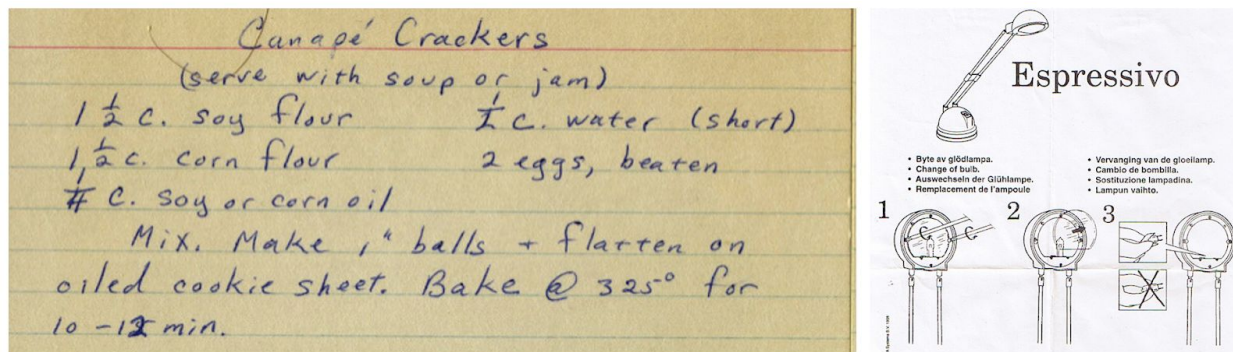
Figure 2. Two pictures depicting a recipe (left) and the instruction for assembling a lamp (right). Left picture by Phil! Gold, source: https://www.flickr.com/photos/phil_g/17282816/. Right picture by Richard Eriksson, source: https://www.flickr.com/photos/sillygwailo/3183183727/.

The word *algorithm* is a combination of the Latin word *algorismus* (that is the Latinization of the name Al-Khwarizmi, who was a great mathematician from Persia in the 8th century) and the Greek word *arithmos*, meaning *number*. Broadly speaking, we can define an algorithm as an

abstraction of a step-by-step procedure that takes something as *input* and produces some desired *output* [Wing, 2008]. Each algorithm is written in a specific language which is functional to communicate its instruction to a computer (either human or machine) so as to obtain something by processing some input material.
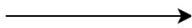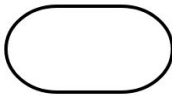
A *computer programmer* usually is a person who creates algorithms and specifies them in a computer program according to a particular computer language – thus, the term computer is here used for talking about an *electronic* computer. However, for what concerns the scope of this course, we use the term computer programmer to refer to anyone that creates algorithms that can be interpreted by any computer, being it a human or a machine.
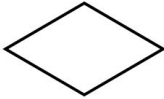
Flowcharts

There is no standard language for describing an algorithm in a way that it is immediately understandable by any computer. However, often Computer Scientists rely on a *pseudocode* when they want to describe a particular algorithm, Broadly speaking, a pseudocode is an informal language that could be interpreted easily by any computer, even if it is usually used for communicating the steps of an algorithm to humans. While an algorithm described by means of a pseudocode is not runnable by an electronic computer, it constructs are closely tied to the ones that are typically defined, with a formal grammar, in programming languages.

In particular, any algorithm can be expressed in pseudocode and, in principle, that pseudocode can be translated into different programming languages quite easily. The real difference is that, usually, some passages in the pseudocode can be simplified by using even natural language text, while one has to specify clearly every passage if one uses a programming language.

In this lecture notes, we use a particular graphical alternative to a common pseudocode which is good for being easily understandable by humans: a flowchart. A flowchart is a particular kind of diagram which can be used to write algorithms, and which relies on a small amount of widgets, as illustrated in Table 1.

Widget	Name	Definition
$\longrightarrow$	Flowline	The arrow is used to define the order in which the operations are executed. The flow indicated by the arrows begin in the starting terminal and ends in the ending terminal (see next widget).
$\bigcirc$	Terminal	It indicates the beginning and ending of an algorithm. It contains a text (usually, either "start" or "end") so as to disambiguate which role has the particular terminal widget in the context of the algorithm.

	Process	It is used for expressing (usually one) instruction or operation, that is executed and that can change the current state of some variables used in the algorithm. The text it includes depicts the instruction to execute.
	Decision	It depicts a conditional operation, where a condition is checked and, depending on the value of some of the variables involved in the algorithm, the execution continues in a particular branch instead of another. Usually, the this operation creates two possible alternative branches: one to be followed whether the condition considered is true, and the other in case the condition is false.
	Input / Output	It allows one to specify possible input / output material which is used / returned by the algorithm usually at the beginning or end of its execution.

Table 1. The main widgets that can be used in a flowchart, and that are useful for writing an algorithm.

The goal of the algorithms we will develop during the initial part of the course must be understood primarily by humans. Thus, a good way for checking if an algorithm one developed (by means of a flowchart) can be interpretable by a computer is to ask a colleague to execute it starting from a particular input – e.g. by writing down all the passages of the execution on a piece of paper.

While such flowchart diagrams can be sketched out on a piece of paper, there exists also online tools that allow you to create a flowchart by using an (electronic) computer. The one that has been used to create all the diagrams in this lecture is called Draw.io, which is a free-to-use Web application with a nice graphical user interface.

Our first algorithm

The goal of today's lecture is to develop our first algorithm. It can be described informally by the following natural language text: taking in input three different *strings*, i.e. two words and a bibliographic entry of a published paper, return *2* if both the words are contained in the bibliographic entry, *1* if only one of the words is contained in the bibliographic entry, and *0* otherwise.

An incomplete version

In the flowchart diagram model, each algorithm is defined by using two terminal widgets, that identify the beginning and the end of the algorithm. The "start" terminal has one arrows starting

from it to the next instruction, while the "end" terminal can be reached by different points of the algorithm, and thus it is linked by at least one arrow.

The first incomplete version of the algorithm, which is introduced in this subsection, simplifies a bit the aforementioned instructions so as to show all the main widget that can be used for creating an algorithm, without adding further complexities. In particular, we want just to say that the algorithm takes in input only two strings, i.e. an input word and a bibliographic entry, and it returns the number *1* if the the input word is contained in the bibliographic entry, otherwise *0* is returned. This partial version of the algorithm is introduced in Figure 3.
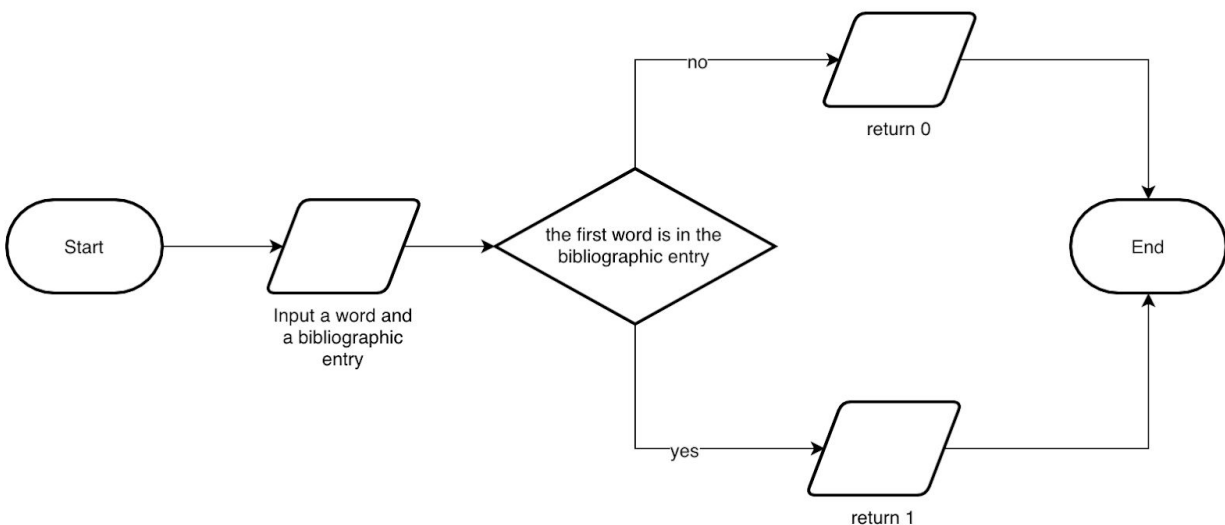

Figure 3. The incomplete algorithm described by a simple flowchart.

In this partial version, several flowchart widgets have been used. In particular, except the "Start" and "End" terminals introduced above, we have used three input / output widgets to get the value specified as input of the algorithm and to returns "0" or "1" depending on such input. The particular decision of what output value to return has been encoded by means of a decision widget, where the input is analysed and, according to the particular situation, the specific "yes / no" branch is followed.

The final algorithm

While the previous subsection introduced a first initial implementation of an incomplete version of the algorithm, in this section a complete version of the algorithm is provided as a flowchart diagram, shown in Figure 4. However, in order to do implement such algorithm with a flowchart, we need to adopt all the widgets introduced in Table 1, by combining a sequence of process widgets with decision widgets so as to implement the whole flow of the algorithm. It is worth noting, though, that the flowchart presented here is just one possible approach to design the original algorithm – other approaches can be indeed used and can be correct as well.

The first of the process widgets is used for initialising the value to be returned. It is executed after the input widget and it prescribes to set a particular variable, i.e. the "result value", to "0", which is the result that the algorithm should return if both of the input words are not contained in the bibliographic entry. This process widget is followed by two sequential decision widgets, that check the two conditions (i.e. whether the first word is contained in the bibliographic entry and whether the second word is contained in it, respectively). In case both the conditions are not passed, then the result value is returned as it has been set originally in the first process widget. Otherwise, every time one of the conditions is passed, the result value is incremented by "1". These passages are responsible for the implementation of the algorithm as requested at the beginning of this section.
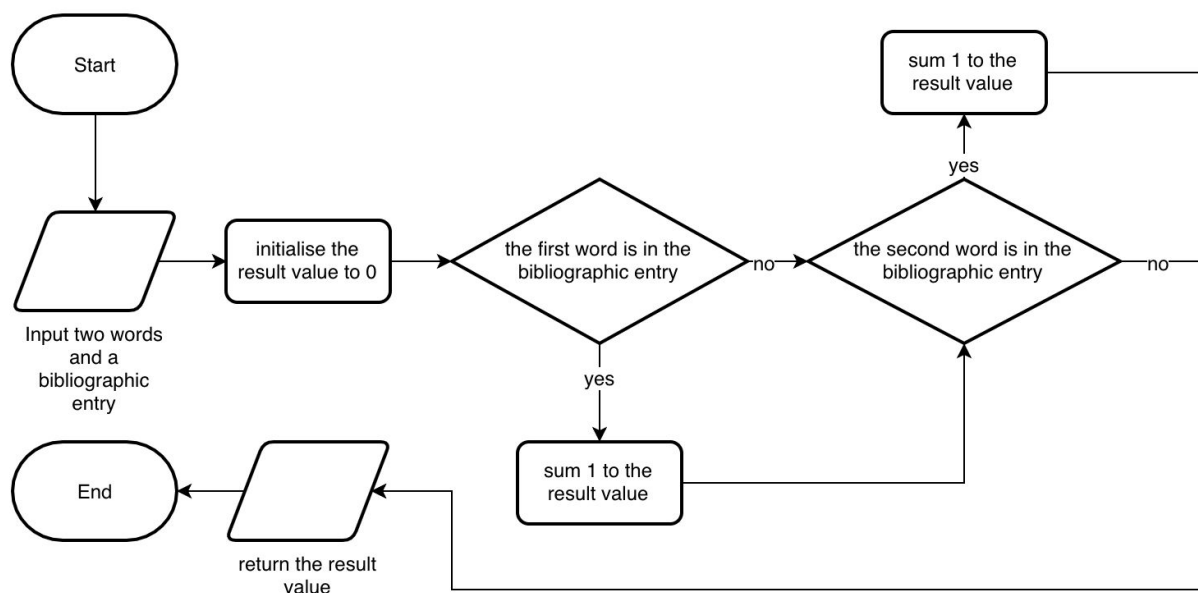
Figure 4. The complete algorithm presented with a flowchart.

Exercises

1. What is the result of the execution of the algorithm in [Figure 4](#) using `"Peroni"`, `"HTML"`, and `"Peroni, S., Osborne, F., Di Iorio, A., Nuzzolese, A. G., Poggi, F., Vitali, F., Motta, E. (2017). Research Articles in Simplified HTML: a Web-first format for HTML-based scholarly articles. PeerJ Computer Science 3: e132. e2513. DOI: https://doi.org/10.7717/peerj-cs.132"` as input values?
2. Write the flowchart of an algorithm that takes in input two objects and returns "yes" whether the two objects are the same, otherwise it returns "no".

3. The previous lecture notes, entitled "Introduction to Computational Thinking", illustrate two different algorithms, expressed in natural language, for implementing the Fibonacci function. Create two distinct flowcharts so as to implement both of them.

References

Menabrea, L. F. (1842). Sketch of the Analytical Engine Invented by Charles Babbage – With notes upon the Memoir by the Translator: Ada Augusta, Countess of Lovelace. Scientific Memoirs, 3. http://www.fourmilab.ch/babbage/sketch.html

Morais, B. (2013). Ada Lovelace, the First Tech Visionary. The New Yorker. https://www.newyorker.com/tech/elements/ada-lovelace-the-first-tech-visionary (last visited 2 November 2017)

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366 (1881): 3717. https://doi.org/10.1098/rsta.2008.0118