Algorithms

Author(s)

[Silvio Peroni](#) – silvio.peroni@unibo.it – https://orcid.org/0000-0003-0530-4305

Digital Humanities Advanced Research Centre (DHARC), Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

Keywords

Algorithm; Ada Lovelace; Flowchart; Pseudocode

Abstract

These lecture notes introduce the notion of *algorithm* and *pseudocode*, to provide the initial tools for instructing a computer in executing a particular task. Also, it presents a specific kind of graphical pseudocode, 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 on 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](#). She was an English mathematician who became famous for her work on Babbage's [Analytical Engine](#). Despite her father's habits, her mother, [Anne Isabella Milbanke](#), actively promoted Ada's interest in logic and mathematics, even after the death of her father. One of her mother's goals was to prevent her from incurring in the same insanity that characterised her father's life. However, Byron's creativity manifested unpredictably anyway.

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 Bernoulli numbers [Menabrea, 1842]. This description was the first computer program ever written: the first implementation of a mechanical computer *algorithm* of the whole history. Ada created it without even having the real implemented machine – 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 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, people have recognised Ada Lovelace 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. The actual goal of the two examples is different: the first is a recipe while the other is a set of instructions for assembling a utensil. However, they are described in terms of a shared abstract notion. Indeed, they are instructions for *producing something* starting from some *initial material* we have, i.e. an algorithm.
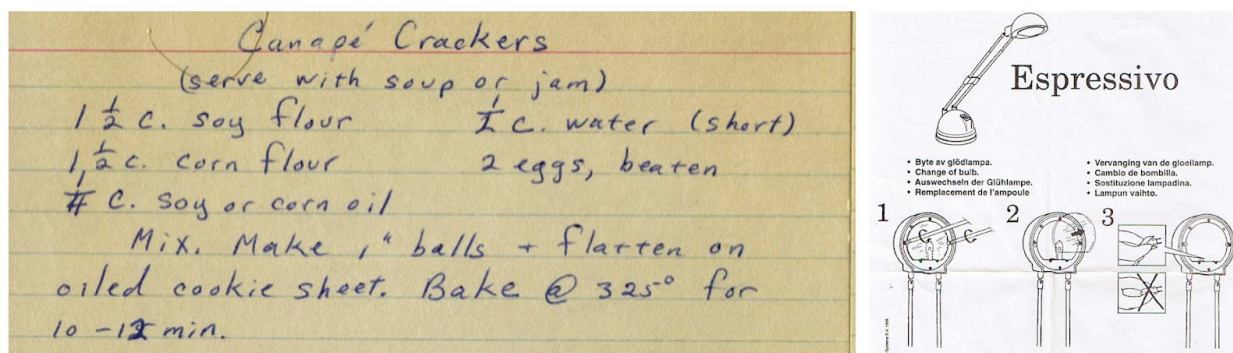
Figure 2. Two pictures depicting a recipe (left) and the instruction for assembling a lamp (right). Left picture by Phil! Gold, source: https://www.flickr.com/photos/phil_g/17282816/. Right picture by Richard Eriksson, source: https://www.flickr.com/photos/sillygwailo/3183183727/.

The word *algorithm* is a combination of the Latin word *algorismus*[1] and the Greek word *arithmos*, meaning *number*. 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. We use such language to communicate the instructions defined by the algorithm to a computer (either human or machine) to obtain something by processing some input material.

Usually, a *computer programmer* is a person who creates algorithms as (electronic) computer programs written in a particular computer language. However, in this text, we use the term computer programmer to refer to anyone that creates algorithms that can be interpreted by any computer, be 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 *pseudocode* when they want to describe a particular algorithm. Pseudocode is an informal language that could be interpreted easily by any computer, even if humans usually use it for communicating the steps of an algorithm to other humans. Often, an algorithm described in pseudocode is not runnable by an electronic computer. However, its constructs are closely tied to the ones that are typically defined, with a formal grammar, in programming languages.

Any algorithm can be expressed in pseudocode. And, in principle, that pseudocode can be translated into different programming languages. Usually, using pseudocode enables one to simplify some passages of the algorithm by describing them in natural language. This simplification is not possible using and proper programming language where one has to specify clearly and formally every passage of the algorithm.

In this lecture notes, we use a particular graphical alternative to common pseudocode which is suitable for being easily understandable by humans: a flowchart. A flowchart is a specific kind of diagram which can be used to write algorithms, and which relies on a small number of widgets, as illustrated in Table 1.
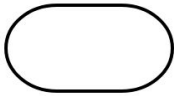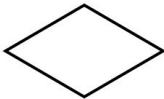
The goal of the algorithms we will develop during the first part of this text must be understood primarily by humans. Thus, a good way for checking if an algorithm one developed can be interpreted correctly by a computer is to ask a colleague to execute it using a particular input. For instance, such execution can be stored by writing down all the passages of the execution on a piece of paper.

---

[1] The word *algorismus* is the Latinization of the name Al-Khwarizmi, who was a great mathematician from Persia in the 8th century.

While such flowchart diagrams can be sketched out on a piece of paper, there also exists 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 friendly graphical user interface.

Widget	Name	Definition
	Flowline	The arrow is used to define the order to follow to execute the operations. The flow indicated by the arrows begin in the starting terminal and ends in the ending terminal (see next widget).
	Terminal	It indicates the beginning and end of an algorithm. It contains a text (usually, either "start" or "end") to disambiguate which role has the particular terminal widget in the context of the algorithm.
	Process	It expresses (usually one) instruction or operation. Its execution 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: it checks a condition, and the execution continues in a particular branch of the flowchart depending on the current status of the algorithm's execution. Usually, this operation creates two possible alternative branches: one to be followed whether the condition is valid, and the other in case the condition is not valid.
	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 widgets that can be used in a flowchart, and that are useful for writing an algorithm.

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:

Consider three different strings as input, i.e. two words and a bibliographic entry of a published paper. The algorithm must return: the number 2 if the bibliographic entry contains both words; the number 1 if the bibliographic entry contains only one word; the number 0 otherwise.

A methodology for the development of an algorithm must focus on specific aspects of the description provided. First, it is crucial to identify the *input material* that such algorithm uses for producing something. In the previous example, the input includes a *first word*, a *second word*, and the *bibliographic entry*, all of them defined as strings, i.e. sequences of characters.

The second important aspect is the *output* that the algorithm should return starting from the input material mentioned above. In the previous example, the output is a number: *2*, *1*, or *0* depending on the particular input that will be processed.

It could be helpful to clearly identify such input and output directly on the natural language description of the algorithm to implement. For instance, considering again the description mentioned above, the following excerpt highlights the input in bold blue text, while the output in italic red text:

> Consider three different strings as input, i.e. **two words** and **a bibliographic entry** of a published paper. The algorithm must return the *number 2* if the bibliographic entry contains both words; the *number 1* if the bibliographic entry contains only one word; the *number 0* otherwise.

In addition to such visual identification of the input and output, it is important to understand which output should be returned by the algorithm according to different input values. The idea is to emulate the execution of an algorithm on specific input by following the informal instruction provided in the natural language description. This operation allows one to understand what it should be the expected result of the algorithm execution before having a concrete implementation of the algorithm (i.e. the flowchart) at hand. For instance:

- input: first word **"Shotton"**, second word **"Open"**, bibliographic entry **"Shotton, D. (2013). Open Citations. Nature, 502: 295–297. doi:10.1038/502295a"** – output: *2*;
- input: first word **"Citations"**, second word **"Science"**, bibliographic entry **"Shotton, D. (2013). Open Citations. Nature, 502: 295–297. doi:10.1038/502295a"** – output: *1*;
- input: first word **"References"**, second word **"1983"**, bibliographic entry **"Shotton, D. (2013). Open Citations. Nature, 502: 295–297. doi:10.1038/502295a"** – output: *0*.

The idea of having such input-output executions is essential to understand the expected behaviour of an algorithm. Besides, these examples can be used to test the flowchart that formally defines the algorithm. For instance, one can use them to understand if all the various instructions depicted by the widgets in the flowchart return the correct output or not.

Finally, it is worth anticipating here the difference that exists between distinct kinds of objects. As shown above, the algorithm must take, as input, three items of a specific type, i.e. *string*, while it always returns another item of another kind, i.e. *number*. Strings and numbers are two distinct data entities. A string is a sequence of characters, and each character can be either a

letter (e.g. "a" and "b"), a digit (e.g. "1" and "9"), a punctuation entity (e.g. "." and ";"), etc. Instead, a number is the usual mathematical entity – *1*, *1983*, *2042*, etc. To distinguish between strings and numbers in the example above, strings are defined between quotations (e.g. "Shotton"), while numbers are represented as such (e.g. 2). Finally, it is essential to notice that strings and numbers are always different entities, even if they may intuitively appear similar. For instance, the string "1983" is not equal to the number 1983 (they are different things, like an orange and a tangerine) even if they may seem similar at first sight.

Given these premises, the next sections will show how to implement the algorithm described in the natural language text above using a flowchart. Let us start.

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 arrow starting from it to the next instruction. One can reach the *end* terminal from different points of the algorithm, and thus it is linked by at least one arrow.

This subsection introduces the first incomplete version of the algorithm. Here the instructions mentioned above are simplified a bit. In particular, we want to say that the algorithm takes as input only two strings: an input word and a bibliographic entry. The algorithm returns the number 1 if the bibliographic entry contains the input word; otherwise, it returns the number 0. Figure 3 shows the flowchart implementing this incomplete version of the algorithm.
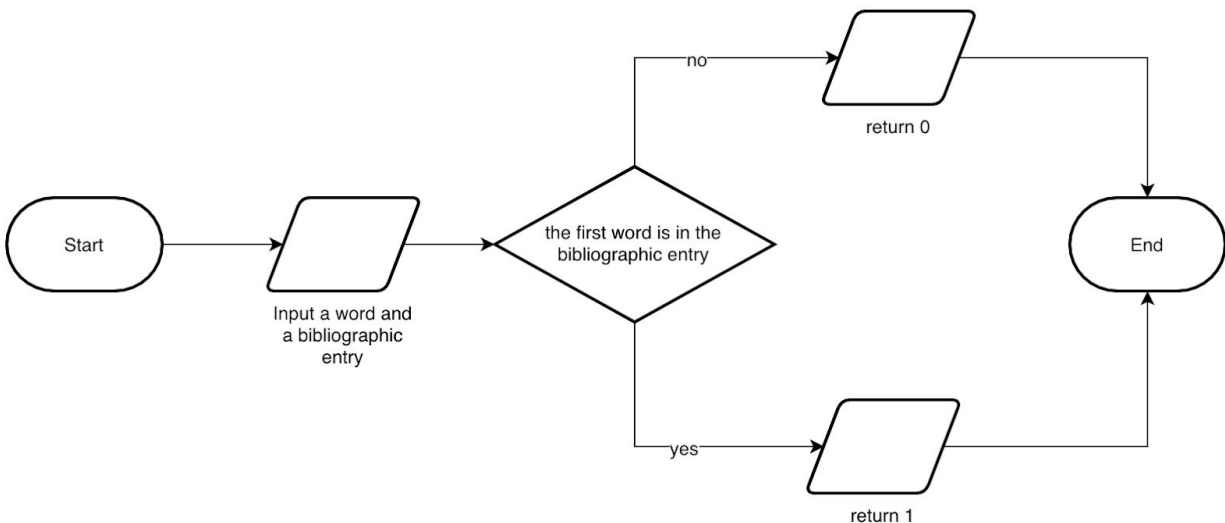
**Figure 3.** The incomplete algorithm described by a simple flowchart.

In this partial version, we used several flowchart widgets. In particular:

- the *start* and *end* terminals introduced above;
- three input/output widgets to get the value specified as the input of the algorithm and to return a 0 or a 1 depending on such input information;
- a decision widget to choose which of the output values to return according to the particular situation.

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 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. Of course, the flowchart presented is just one possible way to design the original algorithm. Other approaches can be indeed used and can be correct as well.

We use a process widget for initialising the value to be returned. We specify it after the input widget, and it prescribes to set a particular variable, i.e. the *result value*, to 0. Such a variable is used to store the result that the algorithm should return, and it will be modified depending on the particular operations executed. The decision widgets check two conditions: whether the bibliographic entry contains the first and the second word, respectively. In case both the conditions are not passed, we return the *result value* as set initially in the first process widget. Otherwise, every time one of the conditions is valid, 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.
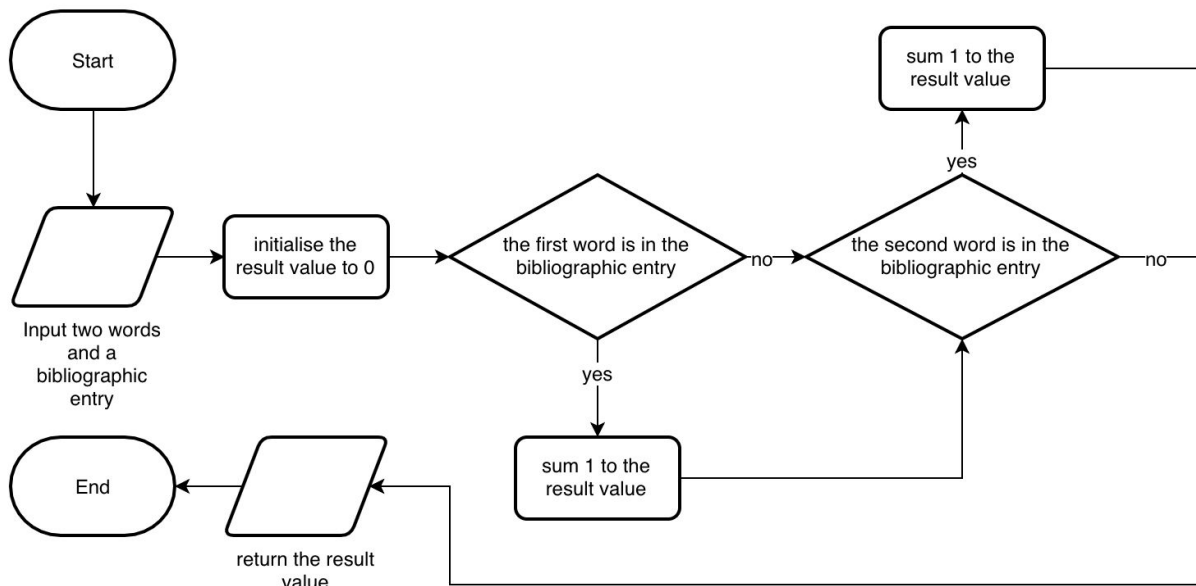
Figure 4. The complete algorithm presented with a flowchart.

Once developed the flowchart, we can check if it works correctly by executing its instruction on particular input data. To this end, we can use the emulated input-output scenarios introduced at the beginning of this section. If the result returned by executing the flowchart with those input values is not equal to the expected output, then it is probable that there is a mistake in the 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 the string "yes" whether the two objects are the same; otherwise, it returns the string "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 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 (last visited 15 October 2019)

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 15 October 2019)

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. DOI: https://doi.org/10.1098/rsta.2008.0118 - also available at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2696102/ (last visited 15 October 2019)