Teaching Algorithms

Date Created November2020

Reviewed

Version

Description

Notes from the Raspberry Pi Foundation with spelling corrected :)

Teaching algorithms

Understanding algorithms ourselves is one thing, but how do we educate learners about them, their importance, and how to implement them as programs? In this section, we'll focus on developing students' understanding of algorithms, and their ability to follow examples as well as write their own.

Working out how to engage students with the concept of algorithms, and encouraging them to write and evaluate their own, is challenging, and the approach you take will depend on the setting, your students, and your own personal confidence. Luckily, there are plenty of ideas and resources available to help you.

Here we outline some of the strategies you might use, with some examples.

Analogy

A significant challenge when teaching computing concepts in general is the fact that they are usually fairly abstract. This makes it difficult for learners to construct their own mental model of a concept. Analogies and stories are a great way to help make an abstract concept more concrete, to give it context and ground it in something familiar. The features of the familiar analogy can then be related back to the original concept.

This journey (called a **semantic wave**), from abstract concept to concrete example and back again, is what makes this technique successful. However it's not without its pitfalls. A successful analogy is a close approximation of the concept to aid understanding. A bad or misinterpreted analogy can introduce or reinforce misconceptions. It's therefore important to:

- Choose your analogies carefully. Use a scenario that is familar to the learners and comparable to the concept being addressed.
- Identify ways in which the analogy is not exact, and the misconceptions that these might lead to.
- Connect the concrete analogy back to the abstract concept. Completing this journey ensures that learners are focused on the concept rather than the analogy, and provides an opportunity to challenge misconceptions.

What stories or analogies might we use to explain aspects of algorithms?

Invention and discovery

There are some concepts that learners may not initially see the value of - concepts that they could do without. For example, skills such as iteration and using functions are things they could manage without and still write successful programs. Rather than simple teaching learners **how** loops or functions work, a teacher might take the approach of helping learners discover or invent these

concepts for themselves. Connecting the concept with the reason for it existing can help learners identify when to apply that concept, as well as how.

To facilitate this approach, you could provide the students with a very limited instruction set, which they use to solve a series of problems of increasing difficulty. As the challenges get more difficult or more lengthy, learners are likely to get frustrated with the tools at their disposal and have an appetite for better, quicker, easier means of solving the problems. At this point, by either introducing a new concept or allowing learners to suggest what is needed, and even invent its syntax, they should be enabled to develop an understanding of why that concept exists as well as how to use it.

A good example of this approach can be found in an activity from Computer Science Education Week called My Robotic Friends, in which learners program simple cup stacking robots using a limited instruction set. This can also be adapted and contextualised, as seen in an adaptation themed around the exploration of Mars.

What benefits and challenges can you see with a discovery based approach?

Kinaesthetic activities and role-playing

An extension to making concepts concrete with analogies is to explore them using actual physical objects. Students might use an algorithm to construct or manipulate an object using a construction toy. Equally, algorithms might be used to describe an image or diagram which students need to follow, or even design in the first place.

In the same way that early exploration of maths often uses physical manipulations to represent numbers, physical artefacts are often useful for learning about how algorithms work. A teacher might use a group of objects to explore sorting, random dice rolls to explore branching programs, or stacks of chairs to represent a stack data structure.

Role-playing takes kinaesthetic activity one step further, and sees learners acting as part of the machine, program or process. Learners might line up and act out a series of steps from a program, and explore computing concepts as they do so. (My favourite activity for this is a set of cards which can be used to model a program and explore sequencing, iteration, selection, threading, debugging and more).

By acting as parts of the computer, students can be encouraged to see how computers approach problems differently from humans, and connect with the mechanisms that make them work.

Do you have any examples of kinaesthetic or role-playing activities that you have found successful?

Puzzles, games and magic

There are other ways in which we can present algorithms and other computing concepts to our learners. Games and puzzles present an approach which some students will find engaging. For example, there are many games and puzzles with simple enough rules that learners could create algorithms that ensure success. These algorithms could then be implemented in code to create simple games or challenges. Below are some games and puzzles that students could explore algorithmically:

- Nim
- Last Biscuit
- Rock, Paper, Scissors
- Towers of Hanoi
- Noughts and Crosses

Many magic tricks also have an algorithmic approach to them, such as the "Twenty-one card trick" which iteratively sorts cards to move the chosen card to a specific location in the deck. More examples of magic-related activities can be found on the Computer Science For Fun website.

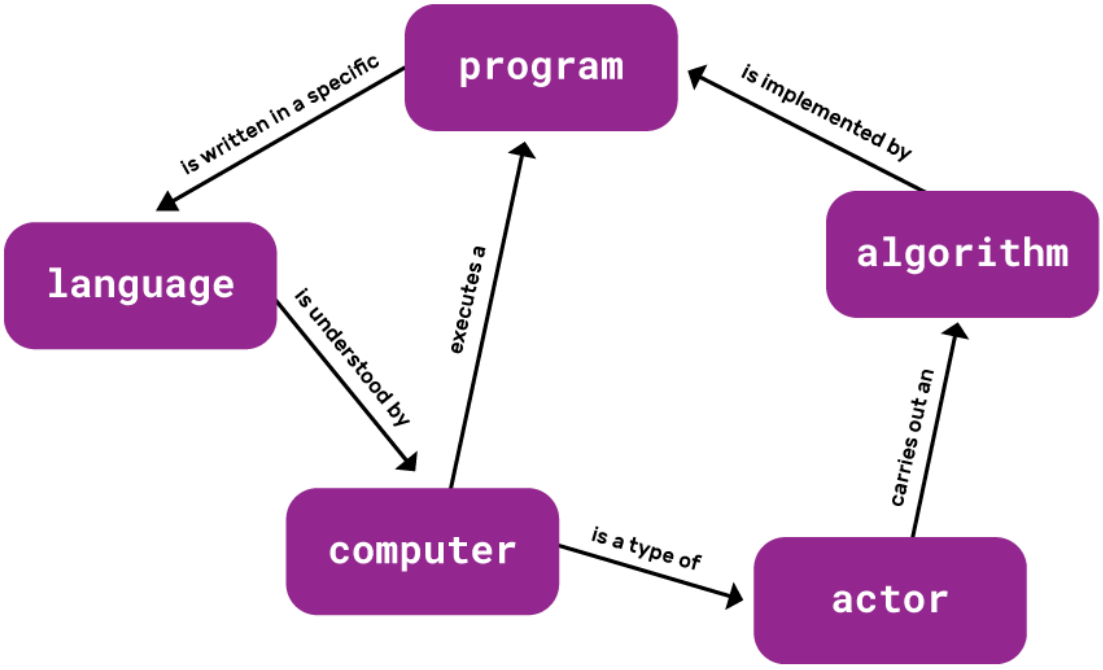
As well as turning existing games into algorithms, there are many programming games online which require students to combine instructions to solve problems. Often these games use a simplified instruction set, usually block-based. Games such as Lightbot and Cargo Bot are popular and certainly involve solving problems using a programmatic approach.

Whilst there is no conclusive research evidence, activities such as these **may** help learners to develop their problem-solving skills and ability to express activities algorithmically. However, as educators we should not rely solely on such activities and should ensure that concepts and misconceptions are explored fully wherever possible. Our role as educators is to connect these scenarios back to their related computer science concepts.

Which activities have you found useful and how do you connect them back to programming?

Mapping Concepts

As in many other subjects, a challenge for learners is how they connect concepts together and build a wider picture of the subject area. This connecting of concepts is important so that learners develop a broader understanding of the subject and see the relevance of concepts in new situations. A concept map can be a powerful way of achieving this, by asking students to record and connect concepts, helping them define relationships and helping educators identify and challenge their misconceptions. An example concept map is shown below:



At what point during the learning would you ask learners to create concept maps?

Summary

Each of the approaches highlighted above represent possible approaches to teaching algorithms and connected concepts. Some will work for you and your learners, whilst others won't. Different approaches may appeal to different groups of learners, and some concepts may need to be addressed using several different techniques.