CT-Nord+TeaEdu

Scaffolding Computational Thinking –

Enhancing Nordic Teacher Education

Module 3

Understanding algorithms and algorithmic problem-solving

DRAFT version, will be finished in June, 2021

Authors: Peter Larsson, University of Turku

Contributors: -

Reviewers: -Valentina Dagienė, Vilnius University

Design (icons): Vaidotas Kinčius, Vilnius University

Module *outline* is based on the work within the project "Scaffolding Computational Thinking - Enhancing Nordic Teacher Education" 2019 (CT-NordTeaEdu). Coordinator: Vilnius University (Lithuania). Partners: KTH Royal Institute of Technology(Sweden), Tallinn University (Estonia), University of Turku (Finland).

The project "Scaffolding Computational Thinking - Enhancing Nordic Teacher Education" (CT-NordTeaEdu) has received co-funding by the Nordplus Horizontal no. NPHE-2019/10157.

Activity 1. Introduction to Computational Thinking and Algorithms

1.1. Lecture: 1 hour 30 min
  1.1.1. Computational Thinking
  1.1.2. Algorithms in mathematics
1.2. Exercises: 60 min
  1.2.1. Find the algorithm in a word problem using CT
  1.2.2. Convert a mathematical procedure to an algorithm
  1.2.3. Create an algorithm to guess a number in a range

Total: 2 hours 30 minuter

Activity 2. Python for algorithms and visualising program structure

2.1. Lecture: 1 hour 30 min
2.2. Exercises: 60 min

Total: 2 hours 30 minuter

Activity 3. Algorithmic problem solving

3.1. Lecture: 1 hour 30 min
3.2. Exercises: 60 min

Total: 2 hours 30 minuter

Activity 4. Complexity or witch algorithm is better

2.1. Lecture: 1 hour 30 min
2.2. Exercises: 60 min

Total: 2 hours 30 minuter

Activity 5. Ethical issues with algorithms

2.1. Lecture: 1 hour 30 min
2.2. Exercises: 60 min

Total: 2 hours 30 minuter

Contribution to the learning outcomes

Learning outcomes	Assessment methods
Understand the PRADA CT-model	Activity 1.2.1 analyse a word problem using CT to devise an algorithm
The algorithmic nature of mathematical procedures	Activity 1.2.2 convert a procedure to an algorithm and Activity 1.2.3 create an mathematical algorithm

Activity 1: Introduction to Computational Thinking and Algorithms

The aim of this activity is to introduce the student to Computational Thinking (CT) and how to use it to create algorithms. Mathematical procedures are used as example of common algorithms.

[The text in this chapter is based on a part "Educational environments for CT: design and aspects of integration" -module authored by Peter Larsson Ashok Veerasamy of University of Turku which were created as a part of TeaEdu4CT Erasmus project. The original module is licensed under CC-BY 4.0 and it has been modified to fit the purpose of the current text.]

Lecture 1.1

1.1.1 Computational Thinking

The PRADA model (Dong et al., 2019) for Computational Thinking (CT) is devised to be a practical way to integrate computational thinking in K-12 education. The aim was to give an understandable picture what computational thinking is. The PRADA acronym consists of Pattern Recognition, Abstraction, Decomposition and Algorithms.

PRADA model (Dong et al., 2019):

- **P**attern **R**ecognition observing and identifying patterns, trends, and regularities in data, processes, or problems
- **A**bstraction identifying the general principles and properties that are important and relevant to the problem
- **D**ecomposition breaking down data, processes, or problems into meaningful smaller, manageable parts
- **A**lgorithms developing step by step instructions for solving [a problem] and similar problems

To recognise a pattern is to use our senses and knowledge to notice that something is recurring or repeating in a systematic way. In mathematics, patterns (Devlin, 1994; Kvasz, 2019) describe the core concepts of a subfield of mathematics by using a small set of facts and rules. From these facts and rules one can derive logically all the other concepts of the field (Devlin, 2012). In computer science (CS) context we use patterns to make explicit the abstract mathematical concepts we are using in our computations. Technically patterns are about controlling the execution of a computer. The execution can have as a goal

calculating (called computing in CS) some mathematical result or getting the computer to do something. In both mathematical and technical view, the first step towards a solution is to recognize the patterns that frame a problem.

We can use mathematics as an example of pattern since it can be said to be the science of patterns. Mathematical patterns can be described as the language we use when we talk about mathematics or mathematical properties of some object of our attention. Without the language, we wouldn't be able to refer to the objects of mathematics and their properties. The mathematical objects and their relationships form the pattern. Given a problem we can use the patterns to recognize if a subfield of mathematics would help in solving it. The pattern is a basis for our calculations and gives us the vocabulary to discuss about the mathematical field. Here we use mathematical theory of numbers better known as arithmetic (see Figure 1.1). to illustrate a mathematical pattern.

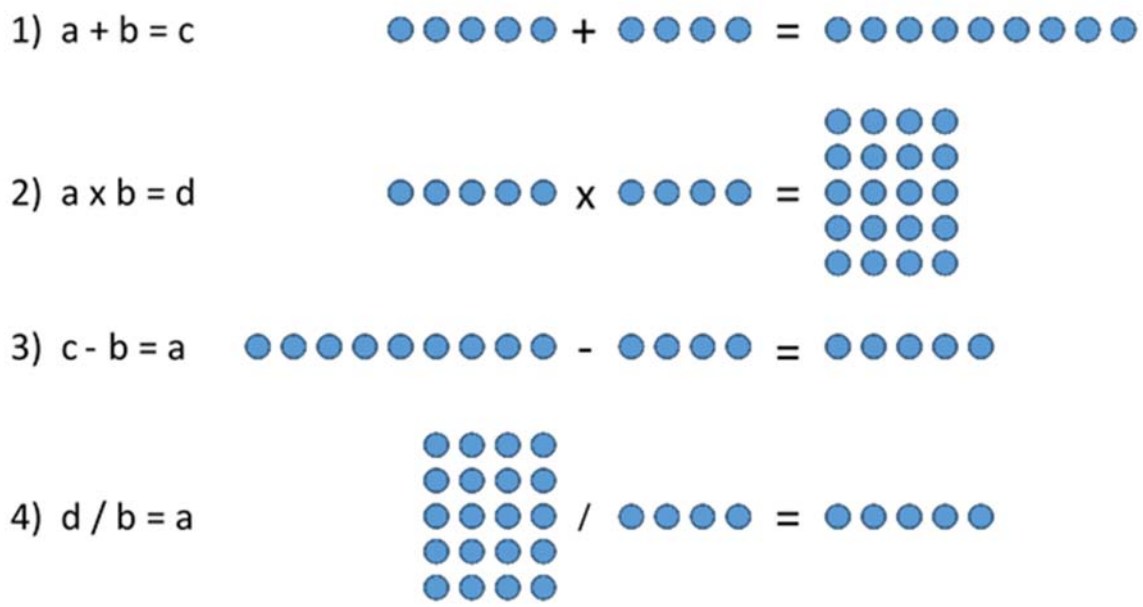

Figure 1.1. Addition, multiplication, subtraction, and division of arithmetic can be visualized to show quantitative patterns related to the symbolic operations.

When we use a pattern as a representative for the original object of our attention then it is an abstraction. Patterns of mathematics, science, engineering, and technology are examples of abstractions. The mathematical objects don't exist in reality, but instead they define regularities that we can use to make sense of our observations. Science uses mathematical models to make exact description of the phenomena of interest. Since a scientific model is usually a combination of basic mathematical equations and is named after the phenomena depicted, we can say that the model is on a higher abstraction level than plain mathematics. In engineering and technology, we use modularity to handle complexity. The name of the modules are abstractions that often describe the function or purpose of a part of a technical structure. In general, to abstract something is to leave out details or group things that are related under a descriptive label.

Mathematics, science, and engineering/technology all employ abstractions. Abstraction is also central concept in CS (Kramer, 2007) and therefore also in CT (Wing, 2006). However, CS/CT requires thinking on multiple levels of abstraction, often two levels at the same time. One example of levels of abstraction used in CS education context is a four-level hierarchy by Perrenet et al. (2005): problem, object, program, and execution (see list below). The levels in the hierarchy are describing different interpretations of an algorithm in programming. At the problem level an algorithm is the change required from inputs to outputs. The inputs are based on the patterns that frame the problem. Object level is thinking about the general steps of the algorithm that are needed to transform the inputs to the outputs. On a program level

the algorithm is implemented in a programming language. The programming language is a high-level description of the operation of a computer or a computational model. The execution level is the physical operation of the computer. Here we have taken the steps from a pattern that is in the context of the problem area to defining an algorithm to solve the problem and its solution as a program executable on a computer. CT is the ability to perform this process.

Abstraction levels of algorithms Perrenet et al. (2005):

- Problem level: change from inputs to outputs
- Object level: algorithm (high-level) description
- Program level: implementation of the algorithm in a programming language
- Execution level: the running of the machine

Decomposition is about how to break down a problem into manageable parts. In CT context the aim is to find an algorithm that would solve the problem. Before decomposition, a problem is only known as a black box with inputs, outputs, and possible relationships to other problems. The black box depicts the innerworkings of the problem that are yet unknown. To find out the inner function, the problem is broken down into multiple subproblems. Each subproblem can be treated as a problem that needs to be unpacked to its component parts. Finally, an atomic level is reached where there are no more subproblems. A solution can be devised based on the identified components of the problem. One can approach the solution top-down, first handling the top-level problems, or bottom-up, first solving the atomic problems.

The decomposition process contains two kind of actions (Rich et al., 2019). The first, substantive decomposition, is to differentiate and categorize the parts (subproblems) of the problem. The second, relational decomposition, is to find relationships between the subproblems. In substantive decomposition one must choose a principle that is used to break the problem in to parts. The principle is dependent on the problem statement and context. The separate subproblems provide information that wasn't accessible while combined. When subproblems have been identified one can start to assign relations between them. Prior to this phase there didn't exist a relation between the problems. Subproblems and their relationships form the pattern of the problem. An example from arithmetic would be the addition of two natural numbers. The relationship between the numbers can be visualized as the difference between positions on a number line. To add one number to another one would take the first number's position as a start and calculate an amount equal to the second numbers position forward. The number of the new position is the result of the addition.

In engineering, CS, design, and other fields, problems are frequently broken down by their functions (Rich et al., 2019). Functional decomposition is a result of the combination of substantive and relational processes. The key to make the decomposition functional is the relationship between subproblems. If the relationship is functional then some operation happens between two subproblems. A functional relationship in arithmetic could be addition, subtraction, multiplication or division. Example of the use of arithmetic functions in other fields of mathematics are the quadratic equation, the greatest common divisor and the Pythagorean Theorem. In CS an algorithm expressed in an imperative type (commanding the computer) of programming language would employ control structures that form the relationship between solutions to subproblems. Breaking down the problem using substantial and relational decomposition helps to gain new meaningful information to devise a (algorithmic) solution (Rich et al., 2019).

Algorithms originated in mathematics and are an important part of it although they are not emphasised. Sometimes any step-by-step process is described as an algorithm, but it is not the process, but the definition that is the algorithm. The definition has to describe the process with mathematical rigour. CS relies on algorithms since its operating principle is based on a computational model. Everything that the computer does is a calculation (called computation), but for the machine to know what to do every action has to be explicit. In a CS context an algorithm can be defined as "…a finite, abstract, effective, compound

control structure, imperatively given, accomplishing a given purpose under given provisions." (Hill, 2016, p. 47).

To decompose Robin K. Hill's (2016) definition of an algorithm (see above) one can start by picturing an algorithm as a definition of a process consisting of discrete steps. An algorithm describes a finite process that must come to an end. The description is abstract and uses only the features needed to depict the change intended. In addition to finiteness the algorithm must be effective which means that the resources and time used must be reasonable. An algorithm is a combination of different control structures that guide the change of its values and finally ends in a result. Control structures and changes to the variables are presented as imperatives using commands. The algorithm should fulfil its intended purpose and nothing else. The algorithm should behave correctly with any input in a specified range.

CT can be interpreted as the mental skills needed to automate the execution of an algorithm (Denning, 2017b). According to the famous computer scientist Donald Knuth (1975; 1985) Computer Science (CS) share algorithms with mathematics. The difference is that in CS we can go beyond what is possible or feasible for a human to compute. However, when we design algorithms in CS, we use mathematics to establish rigour and to calculate if it is possible to execute the algorithm with available computer resources. Since the computer is based on a mathematical model of computation every program is also an algorithm. The world algorithm is however reserved for a generalised solution for a particular type of problem.

1.1.2 Algorithms in mathematics

Algorithms can be introduced by examining the general structure of mathematical procedures since they are based on algorithms. In mathematics, the purpose of an algorithm is to help solve a more complex calculation by dividing it into simpler calculations. Examples of this are paper and pencil calculations, various formulas with instructions such as the Pythagorean theorem or quadratic equation solution formula, and the use of a straight edge and a compass in geometry. The examples above and algorithms in general are only applicable to a particular type of problem. We use our procedural knowledge to execute the algorithms and often we perform them without conscious thinking. However, if would formalise the procedures we would notice that they could be described by calculations performed sequentially, iteratively, and conditionally.

Each algorithm follows a sequential structure and there must be two or more steps (see Figure 1.2). One step would only represent the calculation, so it is not valid as an algorithm. Steps can include calculations or other algorithmic structures. They can repeat things they have done before or make new ones. Towards the end of the algorithm, the results obtained earlier are often compiled.

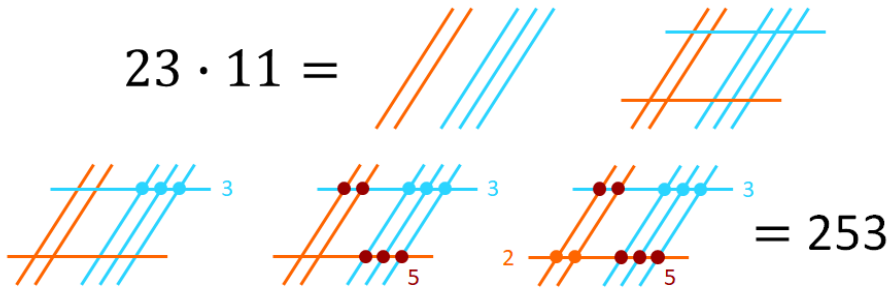

Figure 1.2 A visual multiplication algorithm in which the blue line represents the number one and the orange the number ten. The figure is formed from combinations of blue and orange lines. For multiplication, the first number is shown in vertical lines and the second number in horizontal lines. The result is obtained by summation of the values

represented by the intersecting lines. The value of an intersection is a multiplication of the values the lines represent. The algorithm proceeds sequentially first representing the first and second number as vertical and horizonal lines, then counting the ones represented by intersecting blue lines, then the tens where the blue and orange lines intersect, and finally the hundreds where the two orange lines intersect.

In an iterative structure, a set of successive steps are performed multiple times until some condition is satisfied (see Figure 1.3). This saves you from writing a similar sequence of steps multiple times. By selecting the iteration end conditions appropriately, e.g., as long as there are numbers to calculate or until a value of variable reaches a pre-defined limit, general-purpose algorithms can be created.

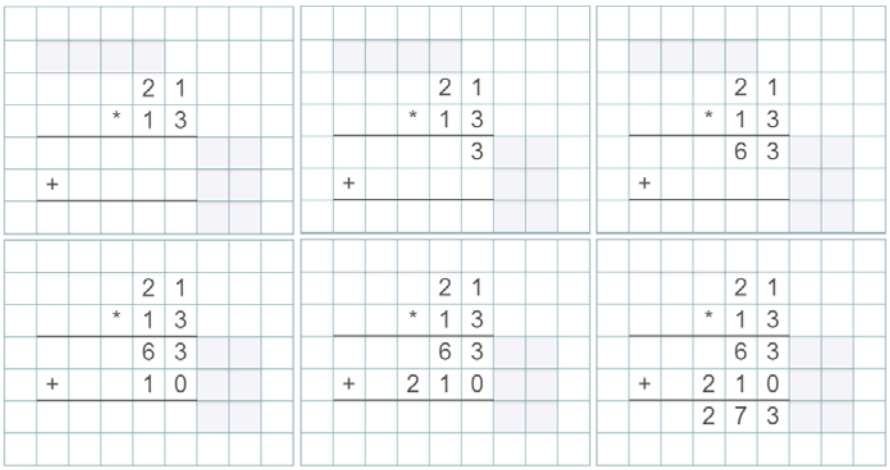

Figure 1.3. Multiplying two two-digit numbers with pen and paper is an example of an algorithm that utilizes repetition. First, the first number is multiplied by the less significant number of the second number and then by the more significant number. In both cases, the procedure is similar, and in this example, the calculations do not require a memory number. The result is two numbers that are finally added together.

The conditional structure allows to define alternative operation depending on the value at hand (see Figure 1.4). The conditional structure may be related to executing a single operation, a choice between two operations, or it may divide the algorithm into two parallel branches. In complex cases the conditional structure can contain several options.

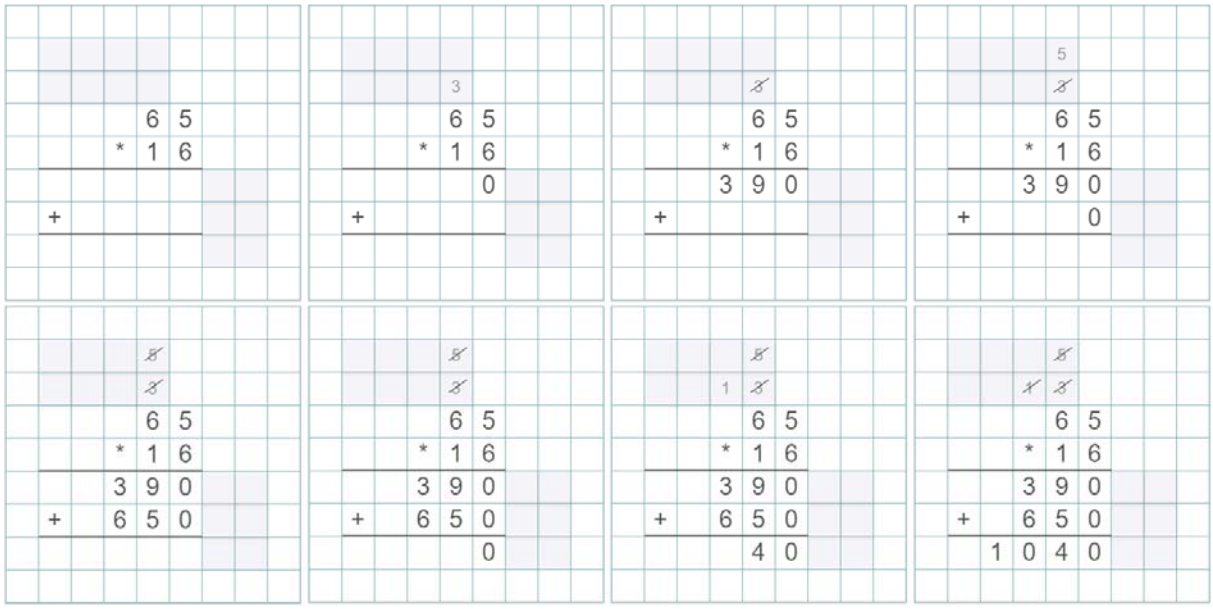

Figure 1.4. When multiplying two numbers using the pen and paper method the carry must be taken into account. That is, if the result of multiplying two digits is more than ten, then the tens are written above the next digit as a carry or if the digits run out over the place where the next digit would be. After the multiplication of two digits the carry is added. The use of a carry can be considered a condition that only takes effect when the multiplication of two digits result in tenths.

The algorithmic structures are simple and when they contain familiar calculations the following of the algorithm is straight forward. However, to design algorithms is a bigger challenge. Two cases can be distinguished: an algorithm for a particular problem and a general algorithm for problems of the same type. For general algorithms, it must be ensured that the algorithm ends in each case and that the solution is correct in all cases. In this module algorithms of both types are introduced.

Exercises 1.2

1.2.1 Find the algorithm in a word problem using CT

Select a word problem (related to mathematics or a topic where mathematics is applied) which is at a level suitable for your students. The aim is to devise an algorithm that solves the problem. The algorithm should be complete and described in a level of detail so that a student could execute it without resorting to outside resources. Use the sequence, iteration, and conditional structures when you describe the order of mathematical operations in the algorithm.

Write small essay where you describe the problem, how you created the algorithm using the PRADA CT-model (see the steps below) and the algorithm you created.

PRADA CT model:

1. Pattern recognition
2. Abstraction
3. Decomposition
4. Algorithm

1.2.2 Convert a mathematical procedure to an algorithm

Choose a mathematical procedure (for instance an equation based on the Pythagorean theorem) which is at a level suitable for your students.

Write an algorithm using mathematical operations and the algorithmic structures (sequence, iteration, and conditional). The algorithm should be able to handle the general cases and if possible, also more special ones. If it is not possible to cover all cases document which kind are an exception and why they cannot be covered.

The algorithm should be complete and described in a level of detail so that a student could execute it without resorting to outside resources.

1.2.3 Create an algorithm to guess a number in a range

Your task is to devise an algorithm to guess a number between two natural numbers. You can ask any question except directly what the number is. You can ask again if your guess is not correct.

The solution should be such that it can be found by just repeating the question and required mathematical operations.

Write the algorithm using mathematical operations, algorithmic structures and a question which can contain variables to refer to the mathematical operations.

Learning resources of the module

Presentations for educators

Videos

Readings for educators and students

Tool for students

References