SUPPORTING ALGORITHMIC APPROACH TO BASIC SCHOOL MATHEMATICS BY PROGRAMMING TASKS

Rein Prank

University of Tartu; rein.prank@ut.ee

The paper describes some findings from a programming task book project. The book contains tasks on arithmetic (operations, calculating the value of numeric expressions, common fractions etc), number theory (factors, prime numbers, GCD, LCM, Euclid algorithm), constructions in planar geometry, algebra (linear equations and inequalities, systems of linear equations, polynomials), some types of nonroutine tasks. The paper also brings up the necessary information processing that is not explicit when mathematical tasks are solved by paper-and-pencil method: parsing of algebraic expressions, finding the coordinates of intersection points by planar constructions, programming of algorithms for the tasks that appear in textbooks only in the form of one single numerical example. Some warnings are given about the impact of "brute force" computer solutions of reasoning-oriented tasks.

Keywords: teaching of programming, factorization, algebraic manipulation, geometry constructions, nonstandard tasks

INTRODUCTION

All developed countries complain about lack of specialists of Information Technology, including programmers. The universities try to take in more IT students and prepare more IT specialists. In order to create the necessary prerequisites, countries experiment with introduction of elements of programming in the school syllabus. Thereby it is quite natural that programming will be taught by the teachers of mathematics (in many countries their training contains certain amount of programming) and even as a part of the mathematics subject. Using mathematics-oriented tasks can help the mathematics teacher to enter the world of programming education. Mathematics textbooks contain many tasks that can be easily reformulated as programming tasks. On the other hand, to prepare more people for IT studies, the algorithmic side of the subject could be brought into greater attention in the teaching of mathematics.

The first major initiative on using programming in teaching of mathematics was LOGO programming (Papert, 1980; Feurzeig & Papert, 2011). LOGO creates a microworld for mathematical experiments. Later the most popular school programming language has been Scratch (Scratch; Benton, Hoyles, Kalas, & Noss, 2016). There exist tens of publications about projects on teaching concrete topics of mathematics using LOGO, Scratch or similar languages. They describe nice programming tasks that are often not just mathematics but can be solved using some mathematical knowledge. Some analyse of learning mathematics through programming is given in (Misfeldt & Ejsing-Duun, 2015). The tasks of our task book are taken directly from the mathematics textbooks. In every chapter of the task book the student should implement or use for solving the tasks the textbook algorithms and solution methods.

Exploring the exercises and theory in textbooks, we see that some problem types are presented together with solution algorithms (like long multiplication or division, operations with fractions, solution of linear equations, some constructions in geometry). However, other (also completely algorithmic) types are presented just with particular numeric examples (Lints, 1981, page 182):

How to measure 7 litres of water using 3- and 5-litre vessels?

One of the possible reasons for this is that the algorithm can be too labour-intensive to execute by hand. In such a case, execution of algorithms on the computer facilitates developing more algorithms for Basic School mathematics. This article describes a project for creating a task book of programming tasks based on the mathematics tasks from the textbooks for grades 1-8 in Estonian schools (age of 7-16 years). The task book formulates programming problems that correspond to different task types in mathematics.

The author has tried to include in the task book not only the most interesting or most important task types but all the task types that have considerable algorithmic character. The resulting task set is the author's answer to the question, What is the algorithmic content of the Basic School mathematics? Unfortunately, this answer is too voluminous for such a short paper. The paper gives a brief overview of the themes of the tasks and concentrates then on the issues that can be not very obvious for the teachers but can be important when using the tasks. We consider some issues of expression manipulation and geometry construction tasks, discuss the possibility of solving reasoning-oriented tasks by "brute force" programming and point to the algorithms for the task types that appear in the textbooks in the form of one single numerical example.

THEMES OF THE TASKS

A large part of the algorithmic content of school mathematics is presented explicitly in the form of algorithms for various types of standard tasks or example solutions. In arithmetic and algebra, the textbooks formulate, for example, algorithms for long multiplication or division, factorization of integers, operations with fractions, solution of linear equations and equation systems, operations with monomials and polynomials. Basic school geometry contains algorithms for bisection of a segment or an angle, for construction of parallel or orthogonal line, construction of a triangle from given three elements. Obviously, we can reword these problems as programming tasks and require writing programs that do the same work. At the time of writing this paper, our collection of mathematics-inspired programming tasks contains 194 items extracted from about 20 Estonian textbooks of different authors. The order of tasks mirrors the places where the underlying mathematical formulation of the task first appears in a textbook for the respective grade. In many cases, the tasks contain more than one variant and some of them can belong to the textbook(s) of higher grade(s). Often the tasks on a mathematical topic begin with a series of preparatory technical tasks. For example, the chapter on geometry constructions begins with drawing segments, rays and lines based on two given points and with drawing circles with a given centre and a given radius or passing through a given point. Some of the subsequent tasks require finding the coordinates of intersection points of two lines, of a line and a circle, and of two circles. Using subroutines for these tasks, it is possible to program school algorithms for bisection of a segment or an angle, for drawing a parallel or an orthogonal line, etc.

The tasks contain computerized variants of tasks from the following topics of the mathematics syllabus:

1. Arithmetic: operations, calculating the value of numeric expressions, decimal digits, equalities and inequalities;
2. Number theory: factors, prime numbers, GCD, LCM, Euclid algorithm;
3. Constructions in planar geometry. Different classes of triangles and quadrangles;
4. Common fractions;
5. Linear equations and inequalities, systems of linear equations;
6. Operations with monomials and polynomials;

7. Solution of some types of nonstandard tasks, using
    a) nested loops,
    b) breadth-first search.

There are also some smaller topics containing (comparatively routine) tasks on

8. Calendar, weekdays, clock;
9. Units of measurement.

Our task book does not contain programming technique exercises that belong to general introductory programming courses. We assume that, before solving our tasks, students have completed a course/chapter on the technical side of programming in some language. The tasks are oriented not to "toy" languages (Logo, Scratch etc) but for "proper" programming (Python, Java, C etc). Working on our tasks also requires more mathematics than studied in grades 1, 2, … that correspond to the underlying textbook tasks. By integration of subjects of programming and mathematics the task book can be used as a source of programming tasks for students. But the teacher can also use a store of implemented solutions for demonstrations or organization of student experiments. Partially or defectively implemented solutions can be used for focusing the attention on special cases in definitions or algorithms.

In the following sections of the paper we discuss the task settings and investigate what additional mathematics and algorithms are necessary for computerized solution of problems in particular topics. Programs for the arithmetic and algebra tasks should combine Basic School mathematics with parsing of expressions. Geometry constructions use the drawing commands. But they are based on coordinates of the points. We establish what amount of analytic geometry would have to be implemented for different construction steps. We describe yet the algorithms for solution of two types of nonroutine problems from the textbooks and analyse the impact of programming on reasoning-oriented tasks.

ARITHMETIC AND ALGEBRA. PARSING AND EXPRESSION MANIPULATION

We discuss here the tasks that belong to computer algebra. The first major topic of our exercises is calculation of the value of arithmetic expressions composed of integers. We start with first-grade exercises containing only one operation, like $2 + 5$, $4 - 1$, etc. The tasks of higher grades contain expressions of growing complexity. In our usual paper and pencil calculations and expression manipulations, we extract decimal numbers, operation signs, parentheses and variables in the expressions almost without formulating explicit rules for this. The textbooks contain explicit rules for the order of operations. Some books can give a list of operation signs and tell that letters of the Latin alphabet represent variables. To solve the same tasks on a computer we should formulate the syntactic rules explicitly, apply some general or task-specific algorithm for parsing the expression, and finally implement a calculation algorithm that solves the actual problem. For learning to use syntactic restrictions we have included tasks with various contents of expressions (different sets of possible operation signs, possibility of unary operations, parentheses). For organization of calculation process we recommend implementing both bottom-up and top-down approaches. Note here also that some programming languages have standard functions (for example, `eval` in Python) that count the value of an expression immediately. For learning/creating the necessary algorithms, our tasks prohibit application of such functions.

The task book contains several chapters with expression manipulation tasks (operations with fractions, solution of equations, inequalities and equation systems, operations with monomials and polynomials). The textbooks present detailed algorithms for solution of many task types and the

programs should just implement them. But the students discover also that some textbook algorithms are not complete. They leave some decisions to the user (e.g., what unknown to isolate first, …). Some algorithms are not described explicitly but should be extracted from a series of examples.

For expression manipulation tasks we use the same expression treatment instruments as in arithmetic but do not need any new tools outside of school mathematics algorithms. The input data of number theory tasks (factorization, prime numbers, GCD, LCM, Euclid algorithm, etc.) consist of one or more integers. The programs for these tasks do not need syntactic supplements. For them it is sufficient to implement just the known mathematical algorithms.

GEOMETRY CONSTRUCTIONS

In the context of mathematics education, programming of geometry construction algorithms means programming of some construction blocks of a dynamic geometry system. The student learns what mathematics is working inside programs like Geogebra. Considering ruler and compass versus computerised solution of elementary mathematics construction problems, we discover that they are based on different information and even on different mathematics. Constructions with paper and pencil belong to Basic School mathematics while writing respective computer programs in general-purpose programming languages requires application of analytic geometry.

Classical ruler and compass constructions do not require any calculations. The intermediate and final results of the constructions are received and used in their graphical form. For drawing a line or a circle we can place the edge of the ruler, or the needlepoint of the compass, at a freely chosen point on a plane, at a point belonging to a curve or at an intersection point of two curves. The task of construction of a point can be completed simply by declaring that the answer is the intersection point of certain two curves.

In programming languages the commands for elementary construction steps (like drawing a point, segment or circle) are based on coordinates, for example, `circle(centre,radius)`. Points, lines and circles can be drawn only when the program "knows" their numerical parameters. This means that information processing of the program that utilises a construction algorithm differs significantly from information processing of the student who makes the same construction on paper. For example, the program for bisection of the angle ABC can consist of the following steps (Figure 1):

1) Draw the circle $c1$ with centre B and radius $R1 = min(|AB|, |CB|)$.
2) Let D be the intersection point of AB and $c1$. Find coordinates of D.
3) Choose the radius $R2 > R1$,
4) Draw the circle $c2$ with centre at D and radius $R2$.
5) Let E be the intersection point of CB and $c1$. Find coordinates of E.
6) Draw the circle $c3$ with centre at E and radius $R2$.
7) Let F be the intersection point of $c2$ and $c3$. Find coordinates of F.
8) Draw the line through B and F – bisector of ABC.

In ruler and compass construction, the calculation steps 2, 5 and 7 are not necessary and the minimum of two radiuses in step 1 can be found on paper without calculating the values of radiuses.

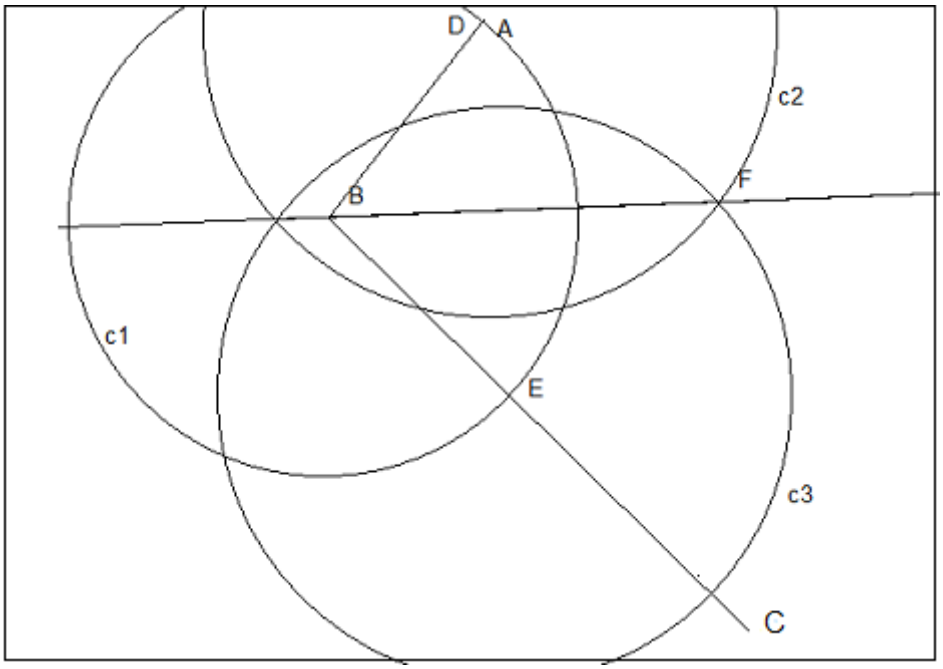

Figure 1. Construction of angle bisector

Note the possibility of an intermediate approach, implementing an interactive program that does not calculate the coordinates of D, E and F but asks the user to point them by mouse (like in dynamic geometry). However, our task book contains only tasks for completely automated solution.

To demonstrate the coordinate-driven character of the drawing commands, we start the chapter on constructions with elementary tasks on drawing of points, segments and circles using different input data (for example, drawing the circle with a given centre and one point on the circle or radius). A particular task can require one or more commands, depending on the programming language. Crucial tasks for modelling ruler and compass constructions are finding the coordinates of intersection point(s) of two lines, a line and a circle and two circles, but also finding the distance between two points. The first task requires solution of a linear system of equations and the fourth uses the Pythagorean theorem. They should be feasible for school students. The second and third tasks are normal exercises in university analytic geometry courses, but their worked solutions should be comprehensive. We have included web links to the theory and detailed mathematical solutions of both tasks so that they can be converted to (branching) programs that find the coordinates. We recommend finalizing the programs for the technical tasks described here in the form of subroutines (functions) that can be used in subsequent constructions. This makes the jump from ruler and compass constructions to their computer implementations easier.

The main goal of our construction chapter is creation of computer programs that model on screen the Basic School ruler and compass algorithms. We saw that, compared to original tasks, this can require additional calculations. The programs should also be capable of handling special cases because the textbook algorithms do not speak about coincidence of points, vertical and horizontal lines, etc. On the other hand, for some tasks the coordinate-based solution can be simpler than doing the same without coordinates. For example, the midpoint of a segment can be received without any construction. We can get the coordinates of the midpoint (as an average) and then use just one command to draw the point. The same can be done for drawing a parallel or an orthogonal line. Together with any task requiring modelling the ruler and compass construction we can ask for a possible easier way to get the result.

REPLACING ASTERISKS OR LETTERS WITH NUMBERS

Many textbooks contain entertainment-style tasks requiring students to replace asterisks or letters with decimal digits and to get a known mathematical structure, for example, a long multiplication scheme as in the left part of Figure 2 (Telgmaa &Nurk, 2002, page 82).

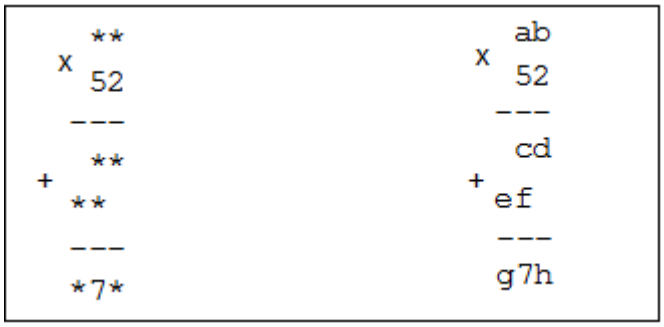

Figure 2. Task of restoring long multiplication

The idea behind such tasks is to make the student analyse what can we conclude about the digits when we take into account the operations between the numbers. For example, our first conclusion can be that the first digit of the first factor should be 1. Otherwise, the first subsum would be a 3-digit number. Further, the first digit of the first subsum can only be 2 or 3 and the second digit of the second subsum can only be 5 or 0. Only the combination of 2 and 5 gives the sum 7, etc.

Consider now what do we get if we ask the student to write a computer program that solves this task. The coarsest approach would be choosing eight variables for the digits replacing the eight asterisks in the figure (Fig 2 right) and writing eight nested loops where the variables $a, \dots, h$ take the values from 1 to 9 for the first digits of the numbers and from 0 to 9 for the remaining variables. For each combination of the values of $a, \dots, h$ the program should check whether multiplication of $10a + b$ with 52 gives the subsums and the final result that correspond to the values of other variables. In reality we only need 7 loops, as $d = h$. The task can be solved already by this brute force approach. Execution of the body of the loop 10^7 times is possible even using a regular laptop. However, the program can be significantly accelerated. The subsums and the product are defined by the values of the factors. We can seek for a and b such that multiplication of $10a + b$ with 52 gives a 3-digit result where the subsums are 2-digit numbers and the second digit is 7. These conditions can be expressed as $52(10a + b) < 1000$, , $2(10a + b) < 100$, $5(10a + b) < 100$ and $mod(div(52(10a + b), 10), 10) = 7$.

For general evaluation of the situation we can tell:

1) The conditions for the digits in long multiplication and other similar schemes are easily expressible in programming languages;
2) The structure of the program is trivial and the program works quickly;
3) Using the loops can be successfully combined/accelerated with using the logically derived conditions;
4) If the student implements only a part of the conditions in the program, then the extraneous solutions can be quite easily recognised and removed.

From this we can conclude that programming is a powerful instrument for solution of tasks of this and similar types. Solving such tasks can persuade students to learn programming. On the flip side, solution by computer allows replacing mathematical thinking with quite routine composing of the loops. Even more, if the student finds a program for some other task of this type, then it would be

quite easy to adapt. In some sense, using computers can ruin a nice type of bonus problems in the textbooks. The teachers must be prepared at least for permanent modification of task conditions.

TASKS WITH HIDDEN ALGORITHMIC ESSENCE

We investigate here another type of nonstandard tasks that can be solved by programming. As early as in the grade 1 textbook (Lints, 1981, page 182), we find the following task:

How to measure 7 litres of water using 3- and 5-litre vessels?

The key to finding the solution is that by filling the 3-litre vessel from a full 5-litre vessel, 2 litres of water remain. Adding it to 5 litres we get 7 litres. There are some other tasks of similar kind in this textbook and in textbooks for higher grades. The textbooks formulate the tasks with concrete numbers of different coins/vessels/… and their sizes. They rely on ingenuity of the brightest students and do not speak about solving such tasks in a general case.

Educated programmers and students who have trained for programming olympiads know that such tasks can be solved using breadth-first search. Every stage in water measuring process can be described by a triplet of numbers (a, b, c) where $a \in \{0, \dots, 3\}$ is the amount of liquid in the 3-litre vessel, $b \in \{0, \dots, 5\}$ is the amount of liquid in the 5-litre vessel and c is the amount of liquid in the vessel for the result. The algorithm finds consecutively the stages that can be reached with 0, 1, 2, etc., steps (pouring operations). The initial stage (0, 0, 0) corresponds to 0 steps. With step 1 we can get (3, 0, 0) and (0, 5, 0). From them we can get with step 2 the stages (0,3,0), (0,0,3), (3,5,0), (3,2,0) and (0,0,5). Further we can construct the stages that need 3, 4, 5, … steps. The task is solved when we get to a stage where the third component is 7. There is no need to construct stages where the third component is higher than 7+5. The search can be finished when at some $n \epsilon N$ no stage gets this number of steps. Our textbook task has two 5-step solutions: (0,0,0), (0,5,0), (0,0,5), (0,5,5), (3,2,5), (3,0,7) or the same with last two stages (0, 0, 10), (3, 0, 7).

The preceding paragraph shows that this task type has a solution algorithm. However, school mathematics does not present such tasks in an algorithmic manner for two reasons. The algorithm uses a data structure (multi-dimensional array) that is too complex for Basic School. Further, the necessary solution time and the required amount of memory grow quickly with increasing input data. The calculation can be executed by hand only if the amount of input data is fairly limited and the number of necessary steps is small. Solving the tasks on a computer has also restrictions of the same kind but the examples from textbooks are computer-solvable. We can demonstrate their algorithmic character. Unfortunately, we must state here again that using computers can change the status of reasoning-oriented tasks.

CONCLUSIONS

Our project of conversion of mathematics tasks to programming exercises gave us about 200 quite natural-looking programming problems. From the perspective of mathematics education:

- Considering mathematical problems as programming tasks is a much better demonstration of the idea of an algorithm. Instead of nondeterministic sets of conversion rules (for operations with fractions or polynomials, for solution of equations and equation systems, etc.), we see these rules being executed in the order that is prescribed by the program. It is possible to change the order and compare different variants.
- Programming of solution algorithms requires a fairly detailed understanding of mathematical theory – definitions and possible special cases, textbook algorithms. It is possible to check this understanding by supplementing the tasks with test data for complex cases.

Many schools and teachers use computer algebra and dynamic geometry software in mathematics teaching. Programming of tasks that correspond to the commands of such programs tells the students what is inside of mathematical software.

- It is possible to emphasize details of algorithms (for example, checking only the numbers up to square root of n by finding the factors of n) and the computational complexity. Note that for paper and pencil calculations this can be more important than for using computers.
- Execution of algorithms on a computer enables them to be applied to much bigger numbers and bigger data. It also facilitates development or at least discussion of solution algorithms for some new types of problems.

Some issues need attention of the teachers.

- In case of certain topics, the program should do some work that is not explicit in paper-and-pencil solutions (understanding of expressions, calculation of coordinates on the plane). This can require some new algorithms and new mathematical knowledge. When supervising such programming, the teacher should have a clear understanding of potential needs that may arise.
- The teacher must know which algorithms have incomplete or example-based formulations available for students in textbooks.
- Some reasoning-oriented tasks, which are quite common in textbooks, can be solved by rather trivial or standard programs. The teachers should be informed about such opportunities.
- On the other hand, programming can help the teachers when they have need to create fresh versions of tasks mentioned in last two sections. Programming enables to evaluate the existence and number of solutions of new versions.

REFERENCES

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2016) Building mathematical knowledge with programming: insights from the ScratchMaths project. *International Journal of Child-Computer Interaction, 16,* 68-76.

Feurzeig, W., Papert, S. A., & with a preface by Bob Lawler (2011). Programming-languages as a conceptual framework for teaching mathematics, *Interactive Learning Environments, 19,* 487-501, DOI: 10.1080/10494820903520040

Lints, A. (1981). *Matemaatika I klassile* [Mathematics for grade 1]. Tallinn: Valgus.

Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. *CERME 9 - Ninth Congress of the European Society for Research in Mathematics Education,* 2524-2530. Retrieved Sept 10, 2019, from https://hal.archives-ouvertes.fr/hal-01289367/document

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas.* New York, NY: Basic Books.

Scratch - Imagine, Program, Share. Retrieved Sept 10, 2019, from https://scratch.mit.edu

Telgmaa, A., & Nurk, E. (2002). *Matemaatika V klassile. 1. Osa.* [Mathematics for grade 5. Part 1]. Tallinn: Koolibri

DOI: 10.17185/duepublico/70746
URN: urn:nbn:de:hbz:464-20191115-140912-1