THE TEACHING OF MATHEMATICS
2022, Vol. XXV, 1, pp. 53–59

TEACHING ABSTRACT ALGEBRA THROUGH PROGRAMMING

Natalia Saealle, Lauri Tart, Indrek Zolk

Abstract. This note presents some examples of programming exercises usable in a first course on abstract algebra. The topics include detecting the properties of an algebraic operation on a finite set, implementing the Euclidean algorithm in different Euclidean rings, long division of polynomials over a finite field, finding all low-degree irreducible polynomials over a finite field.

MathEduc Subject Classification: U55, H45

AMS Subject Classification: 97U50, 97H40

Key words and phrases: Teaching math through coding; algebraic operations; Euclidean rings; Polynomials over a field.

1. Introduction

Teaching math through coding is not a new concept. The first attempts were made already in the 1970s and nowadays there are a number of resources of such exercises available online, e.g., `https://projecteuler.net`, `https://codeforces.com/`, `https://adventofcode.com/`, etc. For enthusiasts, various kinds of programming contests are running; often the exercises are uncodable or the running time is too long if proper preliminary mathematical investigation is not performed. Easier coding-oriented exercises, including math-related ones, are available already from kindergarten up to several levels of educational system, ever since S. Papert began the constructionist movement in mathematics and science education in the 80s [3].

However, in teaching university-level pure mathematics (especially beyond calculus), it seems that programming (in the sense of students writing computer programs in some programming language from scratch) is still rarely used. Computer algebra systems are widespread, though, but there the emphasis is often not on writing new programs but only on inputting mathematical functions and expressions instead [2]. While there have been attempts to bring algorithmic thinking into a university setting, even before personal computers were widely available, e.g., [5], they have been relatively few and isolated.

The main reasons why programming might be rare in pure mathematics courses, could be the following.

- Many topics in pure mathematics deal with sets of too high cardinality and are too proof-oriented for coding. In widespread programming languages it is not possible to work with, e.g., arbitrary real numbers, or with all functions from some set. Of course, using the functional programming paradigm or automatic theorem proving tools, such capabilities are increased but this requires additional knowledge prior to the math course.

- Some students may not have a good command in programming and thus giving (compulsory) programming exercises may be discriminating.
- Professors responsible for pure mathematics courses might not need programming in their research and thus could be reluctant to offer such exercises to students in their courses.

This note focuses on some examples from abstract algebra usable for teaching math through programming.

In University of Tartu, Estonia, starting from Spring 2016, as part of homework of the course "Algebra I" (typically taken in the 3rd semester, compulsory for Mathematics (both major and minor) and Mathematical statistics (major) students but occasionally also taken by Computer Science students), programming exercises have been given to students as part of homework.

The course "Algebra I" itself is a mixture of abstract and linear algebra in a way that the main target is fundamentals of linear algebra (matrices, determinants, linear spaces and subspaces, linear independence, basis, rank of a vector system, systems of linear equations, linear transformations, kernel and range, eigenvalues and eigenvectors, inner product spaces, orthogonalization, orthoprojections, orthogonal and symmetric transformations), but for this, necessary apparatus of abstract algebra starting from groups and up to fields, linear spaces over any field, and polynomials (over arbitrary ring) is developed and studied at full extent.

2. Exercises

The exercises are presented in the following way: a typical example in each category comes first, after that, we describe its variants that we have used in the course.

2.1. Does the algebraic operation have a certain property?

Exercise 1. *Write a program that, given the set* $A = \{1, 2, \dots, N\}$ *and an operation* $*$ *by its Cayley table, decides whether* $*$ *is associative.*

Variants

- The program must decide whether $*$ is commutative.
- The program must decide whether there is an identity element in $(A, *)$.
- The program must decide whether every element in A has an inverse. (The identity element may be given, e.g., 1, or the detection of it may be left to the student.)
- The program must decide whether there is a zero element (absorbing element) in $(A, *)$, i.e., an element z such that for every $x \in A$, there holds $x * z = z * x = z$.
- The program must decide whether there are idempotents in $(A, *)$, i.e., elements x such that $x * x = x$.

- The program must decide whether there are non-zero nilpotent elements in $(A, *)$, i.e., elements x that are not zero and satisfy $x*x*\ldots*x = \omega$ (for some positive number of factors at the lhs) where ω is the zero element of $(A, *)$. (The zero element may be given, e.g., 1, or the detection of it may be left to the student.)
- The program must decide whether there are non-identity unipotent elements in $(A, *)$, i.e., elements x that are not identity and satisfy $x*x*\ldots*x = \varepsilon$ (for some positive number of factors at the lhs) where ε is the identity element of $(A, *)$. (The identity element may be given, e.g., 1, or the detection of it may be left to the student.)

We have used two methods of inputs here.

- Cayley table: the first line of the input file contains N, after that N following lines contain the rows of the Cayley table of $*$, elements are separated by spaces.
- Equalities: the first line of the input file contains N and then N^2 following lines contain the rows in the format `x y z` meaning that there holds $x*y = z$ in $(A, *)$.

We have required the output to go to `stdout`. Usually the output is binary ("yes" or "no"). In the inverse, nilpotent, and unipotent cases it can be ternary (e.g., for inverse detection: "there is no identity"; "the identity is present but a certain element x_0 fails to have an inverse"; "the identity is present and all elements have an inverse").

This exercise seems to be especially good for developing understanding that algebraic operations can indeed be arbitrary and need not have any relation with "ordinary" multiplication or addition. (In written exercises, there is usually not room for defining operations with arbitrary, large Cayley tables.)

Common mistakes

- The data from the file is inputted symbol-wise in a way that every element takes 1 byte. Such a solution might be otherwise correct but is thus restricted only to $N \leq 9$.
- The operation $*$ is interpreted as ordinary integer multiplication at some point of the code instead of using the given Cayley table.
- For two-sided verifications (e.g., $x*e = x \wedge e*x = x$ for identity detection), only one of the equalities is verified.
- For the cases where at least two nested iterations are needed (e.g., commutativity, associativity), the student tries to get the job done with only one iteration.

2.2. Perform the Euclid's algorithm in some "exotic" ring

Exercise 2. *Consider a ring R. The ring R is a Euclidean ring with the following degree function δ. [specific definitions given later]*

The division with remainder in the ring R can be performed in the following way: [description later].

Write a program where two elements a and b of the ring R are inputted, the program performs the Euclid's algorithm using the division described above, and outputs $\gcd(a, b)$.

Settings that we have used

- $R = \mathbb{Z}[\mathrm{i}\sqrt{2}] = \{a + b\mathrm{i}\sqrt{2} \colon a, b \in \mathbb{Z}\}$, $\delta(a + b\mathrm{i}\sqrt{2}) = a^2 + 2b^2$, and in the division step $x = qy + r$ for $x, y \in \mathbb{Z}[\mathrm{i}\sqrt{2}]$ the quotient $q = q_1 + q_2\mathrm{i}\sqrt{2}$ can be found as follows: q_1 and q_2 are nearest integers to rational numbers u_1, u_2 where $\dfrac{x}{y} = u_1 + u_2\mathrm{i}\sqrt{2}$. The hint $\dfrac{x_1 + x_2\mathrm{i}\sqrt{2}}{y_1 + y_2\mathrm{i}\sqrt{2}} = \dfrac{x_1y_1 + 2x_2y_2}{y_1^2 + 2y_2^2} + \dfrac{x_2y_1 - x_1y_2}{y_1^2 + 2y_2^2}\mathrm{i}\sqrt{2}$ can also be useful.
- Same as previous but $\sqrt{2}$ is replaced by 1 (*Gaussian integers*), $\sqrt{3}$ or suchlike.
- $R = \mathbb{Z}[\sqrt{2}] = \{a + b\sqrt{2} \colon a, b \in \mathbb{Z}\}$, $\delta(a + b\sqrt{2}) = |a^2 - 2b^2|$, and in the division step $x = qy + r$ for $x, y \in \mathbb{Z}[\sqrt{2}]$ the quotient $q = q_1 + q_2\sqrt{2}$ can be found as follows: q_1 and q_2 are nearest integers to rational numbers u_1, u_2 where $\dfrac{x}{y} = u_1 + u_2\sqrt{2}$. The hint $\dfrac{x_1 + x_2\sqrt{2}}{y_1 + y_2\sqrt{2}} = \dfrac{x_1y_1 - 2x_2y_2}{y_1^2 - 2y_2^2} + \dfrac{x_2y_1 - x_1y_2}{y_1^2 - 2y_2^2}\sqrt{2}$ can also be useful.
- Same as previous but $\sqrt{2}$ is replaced by $\sqrt{3}$ or suchlike.
- $R = \mathbb{Z}[\omega] = \{a + b\omega \colon a, b \in \mathbb{Z}\}$ where $\omega = -\dfrac{1}{2} + \dfrac{\sqrt{3}}{2}\mathrm{i}$ (*Eisenstein integers*) , $\delta(a + b\omega) = a^2 - ab + b^2$, and in the division step $x = qy + r$ for $x, y \in \mathbb{Z}[\omega]$ the quotient $q = q_1 + q_2\omega$ can be found as follows: q_1 and q_2 are nearest integers to rational numbers u_1, u_2 where $\dfrac{x}{y} = u_1 + u_2\omega$. The hint $\dfrac{x_1 + x_2\omega}{y_1 + y_2\omega} = \dfrac{x_1y_1 - x_1y_2 + x_2y_2}{y_1^2 - y_1y_2 + y_2^2} + \dfrac{x_2y_1 - x_1y_2}{y_1^2 - y_1y_2 + y_2^2}\omega$ can also be useful.

Of course, a typical solution uses either recursion or `while`-iteration until the remainder is zero, and outputs the previous remainder.

This exercise also serves a purpose to show the students different rings and the generality of the concept of Euclidean ring. (If only integers and polynomials are there, the students may obtain a wrong impression that there are no other Euclidean rings.)

Common mistakes

- The allocation of the "previous" remainder is done incorrectly, therefore the program gives wrong answers for trivial cases (e.g., $\gcd(a, a)$, or $\gcd(a, b)$ where $a \mid b$). The same thing can happen for non-trivial cases if the first step of the algorithm is performed partially outside the iteration.
- If `float` data types are used (this is strongly discouraged here!) instead of `int` pairs, there might occur a need to compare, e.g., the "integer" `5e-19` with zero. Similar phenomena can happen if `complex` data type is used where both parts in fact are `float`.

- Using `floor` instead of `round` while finding nearest integers to $u_1, u_2 \in \mathbb{Q}$ can produce infinite loops for certain inputs.

2.3. Perform long division of polynomials

EXERCISE 3. *Let K be a field. Write a program that is, given the coefficients of two polynomials $f, g \in K[X]$ and, performing long division $f = g \cdot q + r$, outputs the quotient q and remainder r (hence, $\deg r < \deg g$). One may assume that $\deg f \geq \deg g$.*

We have used this exercise in the following settings.

- $K = \mathbb{Z}_p$, thus only p is required to be inputted (in addition to f and g).
- K is arbitrary and finite. One can assume $K = \{1, \ldots, N\}$ and the Cayley tables of the operations $+$ and $*$ of K are inputted from files. The student may, of course, assume that $(K, +, *)$ is indeed a field.

Here, the student needs to figure out oneself that taking multiplicative inverse (or solving the equation $\alpha x = \beta$) and additive inverse can and must be done using the arithmetic of K.

Common mistakes

- As it is known, in the long division algorithm, monomials $m_0, m_1, \ldots$ are constructed such that $f = m_0 \cdot g + f_1$, $f_1 = m_1 \cdot g + f_2$, etc, and $\deg f > \deg f_1 > \deg f_2 > \ldots$. Depending on the implementation, it may be troublesome if the degree of a partial remainder f_k is more than one less than the degree of the previous one.
- There have been poorly tested submissions when the program is stuck if f is divisible by g, i.e., when the remainder is zero. (The implementation may look for the highest-degree non-zero coefficient in the partial remainder.)
- If the operations of K are inputted from files, there have been cases where some sub-operation (e.g., finding an inverse, or subtraction) is still performed on $\{1, \ldots, N\}$ as it was arithmetics on integers.

2.4. Irreducible polynomials

EXERCISE 4. *Let K be a finite field. Print all irreducible polynomials $f \in K[X]$ that satisfy $\deg f = 2$ (a harder version: $\deg f = 3$).*

Similarly to the previous exercise, we have used this in two settings: $K = \mathbb{Z}_p$, and arbitrary K whose Cayley tables are inputted from files.

Since any degree zero polynomial (any non-zero constant polynomial) is invertible and any degree one polynomial is non-invertible, there are two obvious ways to solve the (easier version of) exercise.

- Make a list of all polynomials of degree two, and iterate over all pairs of degree one polynomials. Remove these degree two polynomials that are obtained by multiplying two degree one polynomials.

- Iterate over all degree two polynomials f and test all elements $c \in K$. If $f(c) = 0$, then f is divisible by $X - c$ and cannot be irreducible. Those who remain are irreducible.

  For the harder version, either of the ideas can also be adapted.

3. Additional comments

In our course, the programming exercises in homeworks give 1.15% of compulsory credit points of the course. In our situation, the overwhelming majority of students who learn the course, have previously taken at least a programming course in Python (6 ECTS). In some years there have been one or two students from other curricula who have no programming experience; for those we have offered short individual training in Python in the starting weeks.

However, the previous experience of the students with programming varies. Some students program regularly for many purposes, other have limited themselves with minimally passing the compulsory programming course.

The language of programming has been the choice of the student. Since we have Python and Java in first programming courses, these languages are prevailing. Grading means running and looking through the code by the practical lab supervisor. (We do not prefer "black box testing" in learning situations.) Textual feedback is given about the aspects of the student's submission that are malfunctioning.

The students' comments are twofold: either praising that programming exercises were especially interesting and insightful, or disliking such exercises in general. (Our official feedback system, especially giving textual comments, is not compulsory so this is typical that the comments come mainly from those who are very content or are not at all content with some aspects of the course.)

Our beliefs and conclusions on this topic are the following.

- The efforts on the approach *teach (pure) math through programming* should be continued since one of the best ways to understand something is to explain it to somebody and this *somebody* may also be a computer [1, 4].
- Depending on the curriculum, programming in the 2nd year of mathematics bachelor studies can be useful in this respect that the students often learn to program in the 1st semester, but take specialty (e.g., computational mathematics) courses that heavily require it, much later. So there can be a gap to fill in to avoid forgetting programming concepts.
- Clearly there are students who somewhat dislike programming or are poorly familiar with it beforehand, so proper support from the practical lab supervisor must be there to help the students about their homework.

REFERENCES

[1] D. Johnson, *Algorithmics and programming in the school mathematics curriculum: Support is waning – Is there still a case to be made?*, Education and Information Technologies **5**, 9 (2000), 201–214.

[2] M. Okur, R. Dìkìcì, V. A. Sanalan and E. Tatar, *Computer applications in teaching abstract algebra*, International Journal of Applied Science and Technology **1**, 1 (2011), 20–27.

[3] S. Papert, *Mindstorms : Children, Computers, and Powerful Ideas*, Harvester Press, Brighton, Sussex, 1980.

[4] M. Romero, A. Lepage and B. Lille, *Computational thinking development through creative programming in higher education*, International Journal of Educational Technology in Higher Education **14**, 12 (2017), 1–15.

[5] C. C. Sims, *Abstract Algebra: A Computational Approach*, John Wiley & Sons, New York, 1984.

Institute of Mathematics and Statistics, University of Tartu, Narva mnt 18, 51009 Tartu, Estonia

E-mail: natalia.saealle@ut.ee, lauri.tart@ut.ee, indrek.zolk@ut.ee