When there‘s more than one way to get there …

Gabriela Lovászová

Constantinus the Philosopher University
Nitra, Slovakia

glovasz@ukf.sk

Jozef Hvorecký

Vysoká škola manažmentu
Bratislava, Slovakia
&
University of Liverpool
Liverpool, UK

hvorecky@cutn.sk

Abstract

Mathematics is not only about learning problem-solving methods – it is also about gaining a deeper understanding of their purpose, advantages and disadvantages. Frequently, the same problem can be solved by applying several different methods. Mathematics education should also include clues to those best fitting to the person's aim. To achieve that our students are encouraged to find as many correct solutions to problems as possible. Then, properties of individual solutions are discussed.

In the paper we exemplify our approach by solving the problem: How many four-digit numbers not containing 3, 6, and 9 are divisible by 3? *The following methods are shown:*

- *An estimation giving an approximate result,*
- *Constructing a finite automaton recognizing the "divisibility by 3" property,*
- *Forming a formal grammar capable of generating these numbers,*
- *Writing a computer program producing all the numbers,*
- *Using dynamic programming to speed up the execution of the program.*

Comparing different solutions, the students disclose not only what their solutions can do and what they can not do; they are also encouraged to generalize their observations. One of these discussions is also described here.

1 Introduction

Mathematics is more than just a collection of formal problem-solving methods. Mathematicians and qualified users should understand their ***power***: purpose, advantages and disadvantages. Teaching mathematics therefore means more than pure training of calculation techniques and skills – it is also about learning their interrelationships and in-depth understanding of the concepts.

Discussions are an excellent way of building such an intensive kind of knowledge. In our classes, they have a form of "conferences" – students solve problems, present their solutions, compare them with those of classmates and discuss their advantages and disadvantages. As quoted in [5]: *"Conferencing is a medium that can add an extra dimension to developing ideas and increasing understanding of the course material. It gives an opportunity to stop and think and refine ideas ... and hold on to those ideas for future reference."*

The discussions naturally presume students' extensive and intensive knowledge of various methods, both the elementary and advanced one. That's why the theme presented in our paper is taken from the final year of training future teachers of Mathematics and Information Science at the University of Constantinus the Philosopher in Nitra, Slovakia. The course is named *"Selected Topics in Theoretical Informatics"* and addresses contemporary trends in teaching mathematics and informatics. The described goal is one of many. Others are structuring knowledge gained during first years of study, developing teaching skills and capability react in class environment, evaluation of homework, test preparation and evaluation, application of general algorithmic methods and apparatus of formal theories to problem solving.

The course runs for 4 years with the approximate total of 50 students.

2 Teaching Strategy

2.1 Problems

A discussion can evolve only when different ideas are proposed at the same time. In mathematics, such an opportunity appears when a problem can be solved using various methods. Then, the power of each solution can be demonstrated, discussed and generalized.

For our purposes, another feature is also important – the discussed problems must be easy enough for each student to find at least one solution. As we expect all students to participate in the discussion, proposing a solution is the first step towards their higher motivation and involvement. Calling for solutions, we encourage the students to look for „unthinkable", extraordinary solutions. Our experience shows that surprise solutions make later discussions vital and vibrant.

In a way, we are not interested in *solutions themselves* – we expect our students to demonstrate and discuss what the solutions tell *about the problem and themselves*. Therefore, a good argumentation about the selection of the method, its appropriateness, effectiveness and efficiency is a part of its presentation. The students must be ready to "defend" their solutions towards others.

In this paper our method is exemplified using the problem *How many four-digit numbers not containing 3, 6, and 9 are divisible by 3?* Similar discussion raise around *drawing a dragon curve*[1], *robot orientation in a maze*, *generating texts with requested properties, compression and decompression of text strings, creation of fractals* and so on. In average, five themes are analyzed and discussed during the one-term course. Their shared feature is a simple and understandable formulation with many potential solutions.

[1] In [2] the authors describe one of its non-traditional solutions.

2.2 Class activity

To achieve our aims, the process consists of three steps: the preparation of draft solutions, their presentation and, finally, their group reviewing. The last two are the core activities and done inside class in the form of a group discussion. The discussion is moderated as we try to navigate our students to most important observations and relationships. Sometimes – depending on the students' mentality – the presentation and reviewing become mixed up and run simultaneously. We do not insist on their separation unless it does not interfere with aims of the discussion.

The detailed process looks as follows:

1. The problem is given to the students as homework approximately one week before the discussion session. Each student is required to complete at least one solution.
2. The discussion starts with students demonstrating their solutions in the front of their colleagues and explaining reasons of choosing the method. In the later stages, only "distinct solutions" are shown. The student has to indicate in which way his/her solution differs from those presented earlier. This step continues as long as all essentially different solutions are presented.
3. What the class finally gets is a collection of substantially different solutions. Now everyone can express his/her opinion, discuss the appropriateness of the method, its potential improvement, advantages and disadvantages. The remarks pointing to relationships between different solutions are especially welcome and encouraged.

3 Solutions

3.1 Estimating the result

The numbers cannot contain 3, 6, and 9. Seven digits remain. Four-digit numbers cannot begin with 0 i.e. there is one of six digits in the first position and one of seven in all others. Altogether there is 6 x 7 x 7 x 7 numbers not containing "forbidden digits". This can be rewritten as 42 x 49. Approximately, it means 40 x 50 i.e. about 2000 candidates. In general, every third number is divisible by 3. Consequently, our estimation is 667.

3.2 Finite automaton

Figure 1 shows an automaton recognizing whether a number not containing 3, 6, and 9 is divisible by 3. The circles represent their ***states***; the arrows permitted ***moves*** between them. Every evaluation starts in the ***initial state*** q_0. The digits are then read one by one. At every moment, the pair – the input digit & the current state – determines the following state of automaton. Therefore, the first letter determines the second state of the automaton:

- If it is any of the digits 1, 4, or 7, its next state is q_1.
- If it is 2, 5, or 8, then q_2 becomes its next state.

In general, the move from the current state to its successor is specified by the arrow with the input digit. For example, let the current state be q_2:

- When the entered digit is 0, it remains in q_2.
- When the input is 1, 4, or 7, the next state of the automaton is q_F.

- When it is 2, 5, or 8, the next state is q_1.

In fact, the states q_1, q_2 and q_F corresponds to the remainders of the string read so far. The automaton is in the state q_1, when the string gives the remainder 1 after its division by 3. The state q_2 corresponds to the remainder equal to 2. When the state is q_F, the string is divisible by three[2]. The entered number is divisible by 3, if and only if its input process ends in q_F. Therefore, q_F is referred to as the ***final state***. When the input is complete and the automaton is in its final state, the result of the divisibility-by-3 test is "Yes", otherwise it is "No".

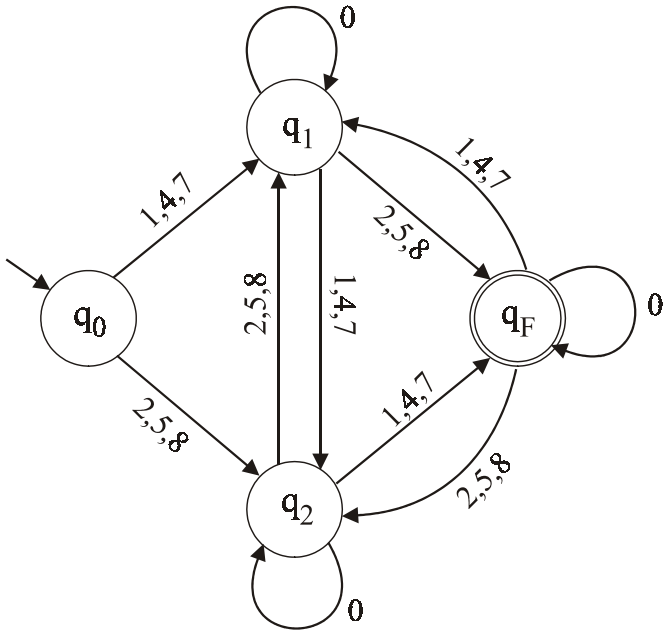

Figure 1. *Finite automaton recognising numbers divisible by 3*

3.3 Regular grammar

Automata help us to recognize whether given string(s) have a given property. Grammars do the opposite – define text strings satisfying a specified property [1]. The regular grammar with the following ***derivation rules*** generates all numbers not containing 3, 6, and 9 and divisible by 3:

$S \Rightarrow 1S_1 \mid 4S_1 \mid 7S_1 \mid 2S_2 \mid 5S_2 \mid 8S_2$

$S_1 \Rightarrow 2 \mid 5 \mid 8 \mid 0S_1 \mid 1S_2 \mid 4S_2 \mid 7S_2 \mid 2S_F \mid 5S_F \mid 8S_F$

$S_2 \Rightarrow 1 \mid 4 \mid 7 \mid 0S_2 \mid 1S_F \mid 4S_F \mid 7S_F \mid 2S_1 \mid 5S_1 \mid 8S_1$

$S_F \Rightarrow 0 \mid 0S_F \mid 1S_1 \mid 4S_1 \mid 7S_1 \mid 2S_2 \mid 5S_2 \mid 8S_2$

The symbols used in the rules belong to two distinct sets:

- The digits – 0, 1, 2, 4, 5, 7, and 8 – are ***terminal symbols***. The generated strings will contain nothing but them.
- All other symbols – S, S_1, S_2 and S_F – are ***non-terminal symbols***. Only non-terminal symbols can appear to the left of the ***derivation symbol*** "⇨" and refer each to other(s) in a recursive way in order to generate sequences of any length.

The derivation symbol indicates that the symbol on its left side can be replaced by the symbol (or the sequence) on its right side. Here, each line is an abbreviation of a

[2] That's why all arrows marked by 0 preserves the current state.

series of right-side strings separated by "|" e.g. S_1 can be replaced either by any of the numbers 2, 5, and 8 or by the sequences $1S_1$, $4S_1$, $7S_1$ etc. Any non-terminal in the current string can be replaced by any right-hand-side sequence[3]. The ***initial symbol*** of the grammar is S so the derivation of every string starts from it. The process terminates when there is no non-terminal symbol in the string.

There are three examples of correct derivations:

$S \Rightarrow 5S_2 \Rightarrow 57$

$S \Rightarrow 8S_2 \Rightarrow 87S_1 \Rightarrow 872$

$S \Rightarrow 4S_1 \Rightarrow 45S_F \Rightarrow 450S_F \Rightarrow 4507S_1 \Rightarrow 45078$

As expected, all three rightmost text strings – the results of the derivations – are numbers divisible by 3. Naturally, to solve the problem we have to generate all and only four-digit strings.

3.4 Non-recursive Computer Program

In our below program written in Pascal, the four-digit numbers are formed as combinations of four elements of the array named *Dig*. Each of its elements contains one of the seven approved digits.

To generate all four-digit numbers, four nested loops are needed. The outermost loop (used for generating the number of thousands) never uses the first element of *Dig* containing zero so the generated number is never less than 1000. The innermost loop combines the selected digits into a newly formed candidate. When it is divisible by 3, a counter (called *Total*) is increased by 1. In the end, *Total* contains the requested number.

```
const Dig:array[1..7] of integer=(0,1,2,4,5,7,8);
var i, j, k, m, Candidate, Total: integer;
begin
Total := 0;
for i:=2 to 7 do
  for j:=1 to 7 do
    for k:=1 to 7 do
      for m:=1 to 7 do
        begin Candidate:= ((((Dig[i]*10+Dig[j])*10+Dig[k])*10+Dig[m])
          if ((Candidate mod 3) = 0)
            then begin inc(Total); Writeln(Total, Candidate) end;
        end;
writeln(Total);
end.
```

[3] The application of the rules with non-terminals prolongs the string.

The program produces the following sequence of results:

Total	Candidate
1	1002
2	1005
3	1008
4	1011
5	1014
6	1017
...	
700	8874
701	8877
702	8880

3.5 Recursive Computer Program

Among many recursive Pascal programs, we have selected one similar to the above finite automaton and regular grammar. It generates the numbers with the required property and stores their number in a global variable named *Total*:

```
var Total: integer = 0;
```

It contains four recursive procedures corresponding to the states of the automaton (and the non-terminal symbols of the grammar). The procedure that simulates the initial state looks as follows:

```
procedure InitialS(Length:integer; S:string);
var New: integer;
begin
if Length > 1 then
     begin
     for New = 1 to 7 step 3 do Remainder1(Length-1, S + New);
     for New = 2 to 8 step 3 do Remainder2(Length-1, S + New);
     end;
end;
```

The argument named *Length* specifies the requested number of digits (four in our case). In each procedure call, it is decreased by 1. The first **for** loop activates the *Remainder1* procedure. It generates numbers giving the remainder equal to 1 by adding digits 1, 4, and 7 to the numerical string *S*[4]. The second loop applies the same to *Remainder2*, by adding 2, 5, and 8.

[4] The digits are placed in the elements Dig[2], Dig[4], and Dig[6], respectively.

The next procedure simulates the state S_1:

```
procedure Remainder1(Length: integer; S: string);

var New: integer;
begin
if (Length = 1) then
     for New = 2 to 8 step 3 do
               begin
                     Inc(Total);
                     writeln(Total, S + New);
               end;
  else
     begin
     Remainder1(Length-1, S + "0");
     for New = 1 to 7 step 3 do Remainder2(Length-1, S + New);
     for New = 2 to 8 step 3 do NoRemainder(Length-1, S + New);
     end;
end;
```

The *then* branch corresponds to the recursion trivial case. As the remainder is 1, it generates three numbers divisible by 3 by adding 2, 5, and 8 to the end of the processed string. The *else* branch generates seven prolongations of its input string:

- When "0" is added, the remainder remains same, so the procedure calls itself.
- When "1", "4", or "7" is added, the remainder becomes 2, so Remainder2 is called.
- After adding "2", "5", and "8", *NoRemainder* is activated.

The *Remainder2* and *NoRemainder* procedures are based on the same idea as *Remainder1*. For space reasons, their detailed formulation is left to the reader.

The execution is activated by the command

```
InitialS(4; "");
```

as our requested number length is 4 and the initial string is always empty. The program produces the same results in a different order:

Total	S
1	1002
2	1005
3	1008
4	1011
5	1014
6	1017
...	
700	8784
701	8787
702	8700

3.6 Dynamic Programming

Dynamic programming is a problem-solving method based on stepwise expansion of the simplest solution into more and more complex ones. For that reason it is frequently expressed by recursive formulas. Recursive solutions can easily be converted into spreadsheet calculations [3]. The table in Figure 2 shows calculations used in our case.

	A	B	C	D
1	Number of digits	Remainder 0	Remainder 1	Remainder 2
2	1	0	3	3
3	2	=B2+3*C2+3*D2	=3*B2+C2+3*D2	=3*B2+3*C2+D2
4	3	=B3+3*C3+3*D3	=3*B3+C3+3*D3	=3*B3+3*C3+D3
5	4	=B4+3*C4+3*D4	=3*B4+C4+3*D4	=3*B4+3*C4+D4

Figure 2 *Formulas used in calculations*

The first row of the calculation corresponds to the trivial case – to the one-digit numbers. As zero is as not assumed to be divisible by 3 here, the value in the corresponding field is 0. There are three numbers giving the remainder equal to 1 as well three giving 2. The formulas calculating the values for longer digits are all of the form

```
= X + 3*Y + 3*Z
```

where X, Y, Z are cells of the previous row. For easier understanding that can be rewritten as

```
= 1*X + 3*Y + 3*Z
```

The field specified as X always refers to the first above field in the same column. It is multiplied by 1, because the sole way of prolonging the numerical string and protect the remainder value is to add 0. Y and Z refer to the remaining two fields in the previous row. Their factor (3) represents three digits that can be added to the shorter strings to produce the new remainder. Figure 3 shows the results of the calculations.

	A	B	C	D
1	Number of digits	Remainder 0	Remainder 1	Remainder 2
2	1	0	3	3
3	2	18	12	12
4	3	90	102	102
5	4	702	678	678

Figure 3 *Results of the calculations*

So, we can conclude that there are 702 four-digit numbers divisible by 3.

4 Discussion

Students' discussions concentrate on the accuracy of the results obtained using the methods, the complexity of their evaluation, and other important features. Students are asked for a full description of each solution and its comparison with his/her other solutions as well as with solutions of their classmates. What follows are some of these observations.

The estimation offers the fastest evaluation. Its disadvantage is its low accuracy. Still, it is an excellent tool for our initial orientation. For example, in the above case the difference is less than 5%. So, when a computer program or a spreadsheet calculation is completed, the estimation is exploited as a yardstick. Significant differences between them are good error indicators.

The finite automaton recognizes whether a number is divisible by 3 or not. One can formally prove that the automaton really performs that – the proof has been informally described above. Unfortunately, the evaluation can be done for one number at time only. To answer our basic question we have test all four-digit numbers manually. Theoretically, it is an excellent solution. Practically, it is a very exhausting one.

One can also prove that the regular grammar generates the correct strings and nothing but them. Again, because we generate a string at time, we have to make many derivations carefully checking whether all strings have been produced, whether no string was generated twice etc. Evidently, it is another impractical method.

The computer programs produce their results quickly. On the other hand, they cannot be implemented without a computer, a programming language – and without a qualified programmer. So, their preparation period can be rather long. Comparing our above programs we see that the non-recursive solution is much easier to complete. On the other hand, because the idea of the recursive program is based on the grammar, it is much simpler to verify that it only generates the specified strings and nothing else. Consequently, it is more elegant. In addition, the recursive program can generate numbers of any length specified by the input, whilst the code of the non-recursive program has to be rewritten to fit the particular length.

The spreadsheet calculation shows the simplest computerized solution to our original question. Compared to other (rather descriptive) ones, this solution looks like a puzzle. It does not generate the numerical strings divisible by 3, it only counts them[5]. Would you guess what it performs, how and why, just looking at Figure 2 and 3? On the other hand, based on our experience with the grammar and the recursive program, the conclusions are much simpler. This also exemplifies why forming a simple and fast solution frequently requires forming less efficient ones first.

[5] To produce the specified numerical strings one has to use an expanding spreadsheet calculation [4].

As a part of discussion, we encourage students to group all presented solutions to "families" – sets of solutions based on the same idea. Two families are presented here:

- The regular grammar, recursive program and the spreadsheet calculation belong to the same family. All are different expressions of the *systematic construction of all and only numerical strings divisible by 3*.
- The finite automaton and the non-recursive program are based on a different idea: *Take all four-digit numbers. Test their divisibility by 3 and count those passing the test*.

5 Conclusions

The methodology allows us to present both complexity of mathematics and interdependence of its elements to students. It allows them to understand that theory-based solutions (finite automata, formal grammars) are excellent to prove the correctness of their ideas, but their direct use is often impractical. On the other hand, they can also see why the use of computer programs and spreadsheet calculations in their isolation from theory is risky. As each calculation is the expression of an idea, the ideas should be based on theoretical reasoning. A proper combination of theory and practice simplifies guaranteeing their correctness.

We see the ability of the discussions to demonstrate strong relationships between mathematical theories and programming practice as the biggest benefit. Also, this active approach to their learning is welcome by the students. Even if we have not made any formal evaluation of the method, their positive attitude to this section of the course and an evident progress in their comprehension of the basic relationships between mathematics and informatics is very encouraging. A student made us happy saying: *I would never guess that mathematics can be an adventure.*

References

1. Hopcroft, J. E. – Ullman, J. D.: **Formal Languages and their Relation to Automata**. Addison-Wesley, Reading, MA, 1969
2. Hvorecký, J. – Lovászová, G.: **Spreadsheets and Languages**. In *Wei-Chi Yang, Sung-Chi Chu, Zaven Karian, Gary Fitz-Gerald (Editors): Proceedings of the Sixth Asian Technology Conference in Mathematics*. : ATCM 2001, Melbourne. pp.371-379
3. Hvorecký, J. – Trenčanský, I: **Recursive Computations in Spreadsheets**. In *Wei-Chi Yang, Kizoshi Shrirayanagi, Sung-Chi Chu, Gary Fitz-Gerald (editors): Proceedings of the Third Asian Technology Conference in Mathematics*. Springer, Tokyo, 1998, pp. 290–299
4. Hvorecký, J. – Trenčanský, I: **Expanding Spreadsheet Calculations**. In *Wei-Chi Yang, Sung-Chi Chu, Jen-Chung Chuan (editors): Proceedings of the Fifth Asian Technology Conference in Mathematics*. ATCM 2000, Chiang Mai, pp. 420–429
5. Salmon, G.: E-Moderating: **The Key to Teaching and Learning Online**. Kogan-Page, London, 2000