Learning by Doing: An Educational Framework for Teaching Combinatorics for Engineers Using Programming Tasks

Doina Logofătu[1] and Manfred Gruber[2]

[1] Frankfurt am Main University of Applied Sciences, Frankfurt am Main, Germany, logofatu@fb2.fh-frankfurt.de
[2] Munich University of Applied Sciences, Munich, Germany, manfred.gruber@lrz.fh-muenchen.de

Abstract

In nowadays teaching of Mathematics the applicative aspect plays an increasingly important role. By joining in teams and using visualisation methods in order to find solutions to given specific problems, learners gain broader abilities and durable mathematical insights. Countable discrete structures, for example, are fundamental for Computer Science and Engineering. Teaching of Combinatorics can be more efficient through programming tasks. It can be taught in a creative and efficient way, by using visual and practical examples, e.g. from arts and chemistry, since there are plenty of such examples around. It provides also an opportunity to exercise programming skills, by writing small but instructive pieces of code for generating, counting or representing specific discrete structures. In our paper we present ways to teach Combinatorics by using programming tasks. For some of them we give implementation examples with C, C++, and Java, and their output, but also use the functional programming paradigm, e.g. with Scheme. In this paper we present an empirical model for teaching Combinatorics, partially tested in different educational modules like Discrete Mathematics, Applied Mathematics, Algorithms and Data Structures, Problem Solving or optional modules like Selected Programming Problems. Besides learning theoretical aspects, the learners will apply instantly their new knowledge and experiment the new terms with different input parameters. They will get in touch and refresh programming skills, work in teams and train their own presentation skills.

Keywords: *Countable discrete structures, applications, learning by doing, competitive programming, experimentation*.

1. Introduction

Mathematics, including its teaching, is largely influenced by its comparably long pre-computer age. When Joseph Fourier, in the early 19th century, developed his analytic theory of heat, observation and measurement were the only sources of experience available. There was nothing beyond he could rely on, except abstraction and reason. There were no means to simulate, visualize and validate his heat transfer model. In nowadays practice, simulation experiments are common in all real-life science or engineering situations. Through simulation experiments scientists and engineers enlarge their experience and gain deeper insights. They can test and validate their models in a riskless way. Experience is important for creativity. Why not base our teaching and learning on more experience? Today's students have grown up with computers. Traditional definition-proposition-proof-lectures do not much inspire them, not even if we add neat elaborate examples to the theory. How can we teach them more inspiringly?

University engineering or computer science programs usually start with Calculus, Linear Algebra and a programming language, followed among others by Discrete Mathematics or an Applied Mathematics courses. The last ones can be taken as good starting points for introducing more explorative learning forms. [13] [18] As an instructor you can choose suitable textbook problems and transform them into programming tasks. You can find suitable programming tasks on the web, too; or you can create programming tasks on your own. You will most likely use an informal but unambiguous style to communicate the task to the students. Almost surely, however, they will complain about a lack of precision in your description. You should not be surprised, on the other hand, if the students' solutions do not precisely deliver what was clearly specified. The solutions to programming tasks are in fact functions assigning prescribed outputs to given inputs. There will follow surely fruitful discussions about the importance of precision and discipline of both definitions and solutions.

Working in teams (three members is a perfect size) leads the students to develop a common language for sharing ideas. Explaining ideas or facts to each other makes them aware of inherent uncertainties in human languages and lets them experience the process of understanding. One can explain only if one has some understanding (or at least the belief to have some understanding). Explaining and discussing math improves (sometimes also corrects) one's own understanding- these are invaluable exercises. Explaining something repeatedly leads to a better knowledge. [17] Students need space to explore and learn. Instructors should not reveal too much about how to find a solution to a given problem. It can be helpful, however, to communicate this simple recipe: if a problem is too difficult to begin with, just simplify - specialize by adding convenient assumptions until you can solve it. Then try to take assumptions back, step by step. Problems are often scalable, not only in university courses, but in real life too.

Usually, discrete mathematics including Combinatorics is taught in the second term, after Calculus and Linear Algebra, but also some Programming classes. In some modules, e.g. Applied Mathematics or Programming exercises would be a good approach to open and adapt teaching to the student's skills and environment. Let them design some demonstrations for some formula (mathematical thinking), or write software programs to solve a specific problem using any language they want and let them continuously describe and present their results to colleagues could boost teaching and make the knowledge last. It would spread their interest, as well research, team work and communication skills. By presentations of learner's approaches and results, students and instructor will be sure the topics were understood. Introducing programming tasks in the mathematical teaching will help students to gain fluency in the use in mathematics for practical problem solving.

2. Topics Selection

Contents and difficulty level have to be adapted to curriculum requirements and students' background. Virtually every mathematical topic can be practiced in such a module, e.g.: Calculus, Linear Algebra, Algorithms Paradigm, Numbers and Graph Theory, Cryptography, Combinatorics, or Probability. Just pick some topics and decide which subjects are appropriate - and to which extent - to meet the students' skills adequately. To begin with, we provide some theoretical background for each topic, introduce and explain the terms used to describe the problem and demonstrate their meaning through simple examples. After that, we let students' teams try to solve as much as possible in the programming language of their choice. The evaluation of their work will be done automatically, either by using test units or an online judge system, e.g. UVa Online Judge [27]. So we can test not only the correctness, but also the run time, of their programs. For untrained students automatic testing can turn out frustrating at the beginning, since specifications have to be met exactly and numerous test cases have to be passed. After a while they will accept that accuracy is unavoidable to survive.

Table 1. Examples for the content of possible contents.

Module 1	Module 2	Module 3	Module 4
Number Theory	Euclidean Geometry	Greedy	Searching
Modular Arithmetic	Numerical Analysis	Divide and Conquer	Sorting
Combinatorics	Simulation Methods	Backtracking	Computational Geometry
Probability and Statistics	Game Theory	Dynamic Programming	Graph Theory

Of course, if the decision is to deepen a specific domain, than you can do less, e.g. only topics of Graph Theory or Simulation Methods.

3. Rules of the Game

Theory and practice sections will alternate during the semester. A theory unit, usually two hours, will be followed by a practice phase. During the practice phase the students work on a problem set, trying to provide correct and efficient programs for the current tasks. They must submit solutions, usually in C, C++ or Java. Alternative, if there is needed; other programming languages e.g. Scala, Haskell, Scheme, are possible by enhancing the judge system. For solutions validation, we can use an Online Judge like, e.g., UVa Online Judge [27] or Sphere Online Judge [26]. A department can also, with some effort, develop a proprietary stand-alone test system, e.g. from students in project modules, or one can individualize a system like DOM Judge. [25] Each team will need an account in the judging system. If a program fails to give a correct answer, the team is notified and can submit another program. The judge gives usually a simple verdict, like e.g. “Accepted”, “Time Limit Exceeded”, “Wrong Answer”, and “Run Time Error”. For the Accepted verdict the team could get also the run time. For the other verdicts they don’t get any additional information. The run time for the Accepted programs is useful if we want to discuss and compare the provided correct solutions. Every team should do some presentations of their programs and have at least two or three feedback discussions/presentations during the semester. These feedbacks will be graded, about 40% of the final grade. At semester end there can be a written final exam or a team presentation about a given topic (about 60%). In this final assessment, the students can provide a feedback about their overall work during the semester.

4. Sample Set of Tasks

In this section we present some typical task examples. Program correctness is implicitly proven by a positive automatic judge (test unit) verdict of the automatically judge (test units) whereas program efficiency can be read off from the run times for various test cases. Usually, unless otherwise specified, the programs communicate through standard input and output. For formatting issues see the sample inputs and outputs. Every task can be generalized and easily converted into a new one, this considerably enriches the discussion session afterwards. Creativity is not only necessary to solve a given task, but also to deal with related problems.

4.1. Task 1 – Circus Night

A number of n ($1<n<11$) couples go to the circus together and sit in a row of $2n$ seats. In how many ways can the $2n$ people be arranged, if each couple sits together (i.e., for each couple, the two people are in adjacent seats).

Sample Input	Sample Output
2	8
5	3840
1	2

Collateral tasks may as well be introduced in discussion: circular permutation – seating at a round table [36]; reduce the condition to sitting in any order, men sit together and women sit together; one couple is arguing and they refuse to sit together, the other couples can sit in any way they want.

4.2. Task 2 – The Book Shelf

For a given number n, $0<n<11$, generate all ways to arrange in a shelf the books coded with the set {1, 2, …, n} in lexicographical order.

Sample Input	Sample Output
3	1 2 3
2	1 3 2
	2 1 3
	2 3 1
	3 1 2
	3 2 1

1 2
2 1

Collateral tasks: provide the number of possibilities [8] [9]; some books are the same in the library, then find the number of possibilities and generate them; number of different words with the letters from one given word, e.g. BANANA or MISSISSIPPI. [1] [9] [16]

4.3. Task 3 – Climbing Man

A man is climbing a stair case, which has *n*, 0<*n*<41, steps. Each time he can either make 1 or 2 steps. How many distinct stepping patterns are there for *n* stairs?

Sample Input	Sample Output
3	3
10	89
19	6765
40	165580141

Collateral tasks: generate a sequence of Fibonacci Primes; Fibonaccimal Base; Fibonacci Polynomial. [32]

4.4. Task 4 – Combinations

Compute the exact number of ways that *n*, 4 <*n*<101, things can be taken *k*, 4<*k*<100 at a time. In this task you have to assume that the final value will fit in a 32-bit C long.

Sample Input	Sample Output
10 3	120
100 94	1192052400
20 4	4845

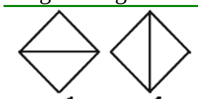
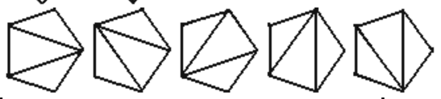
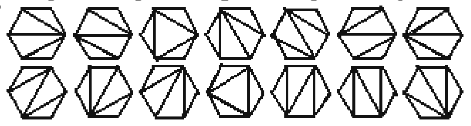
Collateral tasks: Output the number of digits of $\binom{n}{k}$ [28]; output the Pascal Triangle [5][7]; compute the number of derangements. [8] [9]

4.5. Task 5 – Binary Search Trees

Given a number *n*, 0<*n*<1001, find the amount of binary search trees which may be constructed with a set of numbers of size *n* such that each element in the set will be associated to the label of exactly one node in a binary search tree. [UVa, Problem 10303, Wolfram]

Sample Input	Sample Output
3	5
50	1978261657756160653623774456
678	50180443318302554742866846056407722915254622958228602963107290683576122486989615013414022388479003118074063293731486590288146795106568660835607891929825918735675508101151681493705115600538240483988123810933810148626340947846882815525407313572752409154556376772603094344763571687389982044371900504660385185609575311298572827828771485951490269255696808481577238905404506037504917354171611660688847279569760

Collateral tasks: find the number of different binary search trees which may be built with a set of numbers of size *n such* that each element of the set will be associated to the label of exactly one node in a binary search tree [30]; Expression Bracketing [3]; Euler’ Polygon Division Problem. [37]

Regular *n*-gon	Number of Triangulations
	4 sides → 2 triangulations
	5 sides → 5 triangulations
	6 sides → 14 triangulations

4.6. Task 6 – Teams

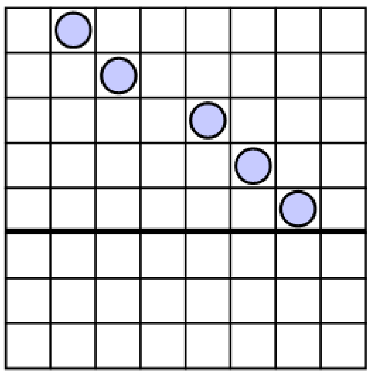
A coach have a total number of *n*, 0<*n*<10^9, players to play a game, selects *k* (1<=*k*<=*n*) players and make one of them as the captain for each phase of it. The task is to find in how many ways a coach can select a team from his *n* players (teams with the same players but having different captain are considered as different teams). For each line of input output the number of ways teams can be selected modulo 1000000007. [31]

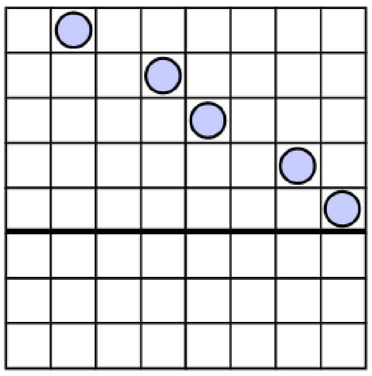
Sample Input	Sample Output
3	Case #1: 5120
10	Case #2: 140615373
237	Case #3: 21554
1234567	

Collateral tasks: calculate the number of digits [9] [28]; solve the same problem, when during the game you must set first and second position in the team can be every time selected; output the last digit of the number. [9]

4.7. Task 7 – Increasing Towers

Generate all configurations of *k* towers which don't attack each other on a chess board of size *nxn* ($4 \le n \le 20$, $1 \le m \le n$), such that they are placed in ascending positions. [9] [10]


Solution: 2 3 5 6 7


Solution: 2 4 5 7 8

Sample Input	Sample Output
6 4	1 2 3 4 1 2 3 5 … 2 4 5 6 3 4 5 6

Collateral tasks: generate all ways to place *n* towers on a chess board of dimension, output the number of these ways; make a minimal change in program to generate *n* queens which don't attack each other on the chees board; generate *k* towers on the first *k* rows chess board with dimension *n*, output the number of these ways. [9] [10]

5. Student activities scenarios, teacher role

Since the student's abilities are diverse, as well as their approaches to deal with the problems, the result will be rich panoply of feedback activities. That's why the teacher's role is important here, since she or he will need to guide the students' work carefully.

After the students got their task set, they realize that they are expected to fulfil numerous requirements. Consequently, a lot of questions arise within the students' teams, e.g.:

1. Which problem shall we solve first?
2. Which programming language shall we prefer for which task?
3. Who takes the responsibility for which part of the problem? When should we better work together, when better alone?
4. After having chosen a task, how do we start?
5. Does there exist only one solution to the problem? Or, if not, how do we find the "best" solution?

If the sequel we offer advice for a subset of tasks mentioned above. For tasks not further mentioned above. For tasks not further mentioned here we refer the reader to our reference section, where more sources are listed. We experienced exciting real class situations, with lots of insights and continuative ideas.

5.1. Task 1 – Circus Night

First, we consider the different ways of arranging the couples themselves, which is a permutation on *n* blocks. Second, within each couple, there two ways to seat individuals. Since there are *n* couples, we multiply two *n* times: $P(n,n)\times\underbrace{2\times 2\times ...\times 2}_{n\ \text{times}} = n!\cdot 2^n$.

Sample C Code Snippet [9]

```c
unsigned long r  = 2;
for(unsigned long i=2; i<=n; i++)
        r *= i*2;
```

Sample Code Snippet Scala [23]

```scala
BigInt(2*2) *
(BigInt(2) to BigInt(n)
   reduce((prev, curr)=> prev*curr*BigInt(2)))
```

Here we can discuss the close formula and some variations (collateral tasks), the efficiency of the code samples and their improvements, how to use recurrence efficiently, static array for factorial. What happens if *n* increases to 100? How many digits will the solution have? What about Scala implementation?

5.2. Task 2 – The Book Shelf

There are lots of algorithms for permutations generation: next permutation, iterative and recursive ones [14].

Sample C++ method [9]

```cpp
int transform_next(vector<short>& P){
  short n = (short)P.size();
  short t, k=n-2;
  while(k>=0 && P[k]>P[k+1]) k--;
  if(k>=0){
    t=n-1;
    while(P[t]<P[k])t--;
    swap(P[k], P[t]);
    reverse(P.begin()+k+1, P.end());
    return 1;
    } else {
```

Sample Java Method (recursive backtracking) [10]

```java
void back(List<Integer> x, int n, PrintStream out){
  int i, xk, k=x.size();
  boolean flag;
  for(xk=0; xk<n; xk++){
    flag = true;
    if(x.size()==k+1) x.remove(x.size()-1);
    for(i=0; flag && i<k; i++)
      if(x.get(i)==xk || Math.abs(xk-x.get(i))==k-i)
        flag=false;
      if(flag){
        x.add(xk);
        return 0;
    }
}
```

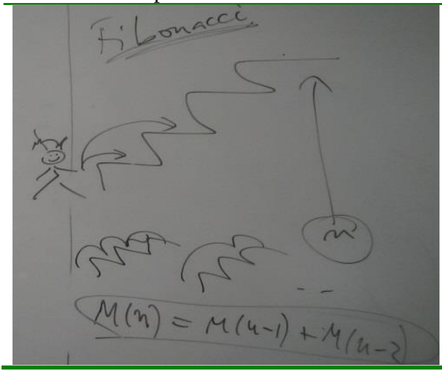
```
                if(k==n-1) writeSolution(x, out);
                else back(x, n, out);
}}}
```

Some suggestions for the discussion: describe some algorithms for permutations generation [14][15][16], compare the run time of the different solutions, ask for minimal changes to generate another structures like permutations with repetition, derangements, permutations ranking/unranking [16], *n*-queens (Java method /blue italic condition)[10].

5.3. Task 3 – Climbing Man

Of course, is easy to see we deal with the Fibonacci sequence. We present here a Scheme [22] implementation.

Whiteboard capture	Sample Scheme Method (naïve binary recursion) [22]



```
(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1)) (fib (- n 2)))))
```

Sample Java Method (iterative) [10]

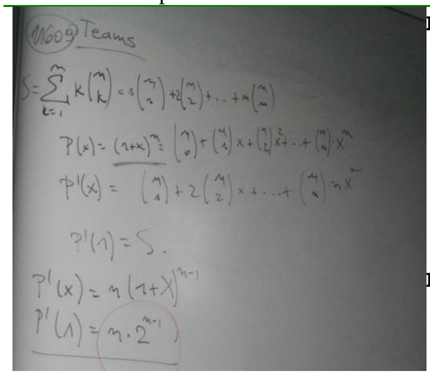
```
private static long fibo2(int n) {
  if (n < 2) { return n; }
  long prevprev = 0, prev = 1;
  for (int i = 2; i < n; i++) {
    long temp = prev; prev += prevprev;
      prevprev = temp;
    }
    return prev + prevprev;
  }
```

The Fibonacci sequence is a very well known, and describes many patterns in nature – some examples, e.g. [5][10]. Using programming of the Fibonacci numbers, the students can easily analyze the efficiency of different paradigm, e.g. the naïve binary recursion algorithm works only for small *n* and is impracticable [5][10].

5.4. Task 6 – Teams

First, we have to find a closed formula for the number of teams. We need as well modular arithmetic knowledge for dealing with the modulo adjustment.

Whiteboard Capture	Sample Java Methods


(11609) Teams
$S=\sum_{k=1}^{n} k\binom{n}{k} = 1\binom{n}{1}+2\binom{n}{2}+\dots+n\binom{n}{n}$
$P(x)=(1+x)^n=\binom{n}{0}+\binom{n}{1}x+\binom{n}{2}x^2+\dots+\binom{n}{n}x^n$
$P'(x)=\binom{n}{1}+2\binom{n}{2}x+\dots+n\binom{n}{n}x^{n-1}$
$P'(1)=S.$
$P'(x)=n(1+x)^{n-1}$
$P'(1)=n\cdot 2^{n-1}$

```
public static long fastExp(long x, long n, long mod){
  long result = 1;                /*fast exponentiation*/
  while (n > 0) {
    if ((n & 1) != 0) {        /*n is odd, bitwise test*/
      result = (result*x) % mod; n-=1;
    }
    x = (x*x) % mod;
    n /= 2;              /*integer division, rounds down*/
  }  return result % mod;
}
public static void main(String... strs) throws
                    NumberFormatException, IOException {
  Scanner sc = new Scanner(System.in);
  long cases = sc.nextInt(), result = 0, input = 0;
  long m = 1000000007;
  for(int i=0; i<cases; i++){
     input = sc.nextInt(); System.out.println("Case
          #"+(i+1)+":  "+(input*fastExp(2,input-1,m)%m));
  }}
```

The board capture shows a demonstration for the number of possibilities; afterwards we need to use some modular arithmetic formula to find out this number modulo 1000000007. [5] [9] An alternative would be to experiment with the Java class *BigInteger*. [20] [21] A presentation of this class would be welcome, since dealing with big numbers could solve easily many practical problems.

7. Conclusion

The experiences with practical team work are numerous, both for students and for instructors. Teams can be more or less successful. Sometimes it is difficult to estimate the amount and value of each individual contribution correctly. Teamwork means extra effort for students and instructors. On the other hand, the benefits cannot be denied. One question often arises: is there one or are there many solutions to the given problem? In many cases there is more than one essentially different solution. If the instructor has specified the goal rather the method, the students' solutions will show a greater variety of possibilities. It can be awarding for all participants to examine and discuss the different solutions in detail. Another question is: what is a good solution? This is sometimes difficult to decide. Even more difficult are the decisions about style. In general it is preferable not to discourage but to encourage. Simplicity and clearness are important and incontrovertible criteria. Finally, it is important to reflect what has been achieved. Solving practical problems is an important activity itself. Being able to solve them is good for the self-consciousness. The next and deeper question, however, has to be: what have we learned? This question leads to the theory behind the problems just solved. The students feedback about this kind of teaching math are unexpected excellent – they finally appreciate they gained lots of knowledge and abilities. Besides, they really have fun!

Referring to Fourier's work, mentioned in the introduction, it is, after practical experiences, hopefully exciting and illuminating to learn more about Fourier's method to solve the heat transfer problem. "No theory without practice!" is only one half of Don Knuth's advice once given to an audience. The other half is: "No practice without theory!"

References

[1] M. Aigner, G. M. Ziegler, K. H. Hofmann, *Proofs form THE BOOK*, Springer, 4*th* Ed., 2010, pp. 159-218.

[2] A. T. Benjamin, J. Quinn, *Proofs that Really Count: The Art of Combinatorial Proof*, The Mathematical Association of America, 2003.

[3] M. K. Brown, C. Hershock, C. J. Finelli, C. O' Neal, Center for Reasearch on Learning and Teaching (CRLT), "*Teaching for Retention in Science, Engineering, and Math Disciplines: A Guide for Faculty* ", 2009, http://www.crlt.umich.edu/sites/default/files/resource_files/CRLT_no25.pdf, Last accessed 1st March 2014.

[4] T. Davis, "Catalan Numbers", 2006, http://mathcircle.berkeley.edu/BMC6/pdf0607/catalan.pdf, Last access 1st March 2014.

[5] R. L. Graham, D. E. Knuth, O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley Professional, 2nd Ed., 1994, Chap. 7-8, pp. 153-319.

[6] D. L. Kreher, D. R. Stinson, *Combinatorial Algorithms, Generation, Enumeration, and Search*, CRC Press, 1999.

[7] G. Haggard, J. Schlipf, S. Whitesides, *Discrete Mathematics for Computer Science,* Thomson Books/Cole, Belmont, 2006, Chap. 7, pp. 421-474.

[8] J.H. van Lint, R. M. Wilson, *A Course in Combinatorics,* 2nd Ed., Cambridge University Press, Cambridge, 2001.

[9] D. Logofătu, *Algorithmen und Problemlösungen mit C++*, Springer Vieweg+Teubner, 2nd Ed., Wiesbaden, 2010, Chap. 7+8, pp. 151-204.

[10] D. Logofătu, *Grundlegende Algorithmen mit Java,* Springer Vieweg+Teubner, Wiesbaden, 2008, pp. 149-239.

[11] D. Logofătu, „Şirul lui Catalan"(Romanian for *Catalan Sequence*), GInfo, 15/5, pp. 36-41, http://www.ginfo.ro/revista/15_5/mate1.pdf, Last access 1 Mach 2014.

[12] G. Polya, *Mathematical discovery: On understanding, learning, and teaching Problem Solving*, Wiley, New York, 1967.

[13] S. S. Sazhin, "Teaching Mathematics to Engineering Students", International Journal Engineering Education, Vol. 14, No. 2, pp. 145-152, 1998.

[14] R. Sedgewick, “Permutation Generation Methods”, Comput. Surveys, Vol. 9, pp. 137-164, 1977.

[15] R. P. Stanley, *Enumerative Combinatorics: Volume 1*, Cambridge Studies in Advanced Mathematics (Book 49), Cambridge University Press, 2nd Ed., 2011.

[16] D. Stanton, D. White, *Contructive Combinatorics*, Springer, 1986.

[17] T. Tao, *“There’s more to mathematics than rigour and proofs”*, 2009, http://terrytao.wordpress.com/career-advice/there%E2%80%99s-more-to-mathematics-than-rigour-and-proofs/, Last accessed 1 March 2014.

[18] A. Y. Vaninsky, “Intuition – Based Teaching Mathematics for Engineers”, ACEEE International Journal on Network Security, Vol. 1, No. 1, pp. 6-11, January 2010.

[19] Java API – Oracle Documentation, http://docs.oracle.com/javase/7/docs/api/, Last accessed 1st March 2014.

[20] C++ Reference, http://www.cplusplus.com/reference/, Last accessed 1st March 2014.

[21] Java API – Oracle Documentation, class math.BigInteger,

http://docs.oracle.com/javase/6/docs/api/java/math/BigInteger.html, Last accessed 1st March 2014.

[22] MIT Scheme Documentation - MIT/GNU Scheme 9.1, http://www.gnu.org/software/mit-scheme/documentation/mit-scheme-ref/, Last accessed 1st March 2014.

[23] Scala Documentation, http://docs.scala-lang.org/tutorials/, Last access 1st March 2014.

[24] ACM-ICPC Live Archive, https://icpcarchive.ecs.baylor.edu, Last accessed 1st March 2014.

[25] DOM Judge – Programming Contest Jury System, www.domjudge.org, Last accessed 1st March 2014.

[26] Sphere Online Judge, http://www.spoj.com/problems/classical/, Last accessed 1st March 2014.

[27] UVa Online Judge, http://uva.onlinejudge.org, Last accessed 1st March 2014.

[28] UVa Online Judge, Problem 10219 – Find the ways!,

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=14&page=show_problem&problem=1160, Last accessed 1st March 2014.

[29] UVa Online Judge, Problem 10303 – How Many Trees?, http://uva.onlinejudge.org/external/103/10303.pdf, Last accessed 1st March 2014.

[30] UVa Online Judge, Problem 10007 – Count the Trees, http://uva.onlinejudge.org/external/100/10007.pdf, Last accessed 1st March 2014.

[31] UVa Online Judge, Problem 11609 – Teams, http://uva.onlinejudge.org/external/116/11609.pdf, Last accessed 1st March 2014.

[32] P., Chandra, Eric. W., Weisstein, “Fibonacci Number”, From *MathWorld*—A Wolfram Web Resource,

http://mathworld.wolfram.com/FibonacciNumber.html, Last accessed 1st March 2014.

[33] Eric. W. Weisstein, "Binary Tree." From *MathWorld*--A Wolfram Web

Resource. http://mathworld.wolfram.com/BinaryTree.html, Last accessed 1st March 2014.

[34] Eric. W. Weisstein, “Bracketing.”, From *MathWorld*—A Wolfram Web Resource,

http://mathworld.wolfram.com/Bracketing.html, Last accessed 1st March 2014.

[35] R. Stanley. Eric W. Weisstein, "Catalan Number." From *MathWorld*--A Wolfram Web

Resource. http://mathworld.wolfram.com/CatalanNumber.html, Last accessed 1st March 2014.

[36] Eric W. Weisstein, "Circular Permutation." From *MathWorld*--A Wolfram Web

Resource. http://mathworld.wolfram.com/CircularPermutation.html, Last accessed 1st March 2014.

[37] Eric W. Weisstein, "Euler's Polygon Division Problem." From *MathWorld*--A Wolfram Web

Resource. http://mathworld.wolfram.com/EulersPolygonDivisionProblem.html, Last accessed 1st March 2014.

Authors

Principal Author: Doina Logofătu holds a PhD degree in Computer Science from Babes-Bolyai University in Cluj-Napoca, Romania. She worked in the educational sector, as well in the industry. She wrote several books about programming and mathematics in Germany and Romania. At present she is holding a professorship for mathematics and computer science at Frankfurt am Main University of Applied Sciences.

Co-author: Manfred Gruber holds a PhD degree in Mathematics from the Bayreuth University, Germany. He worked in industry for many years and since 1991 in the educational sector, as Professor for Mathematics with Nurnberg University of Applied Sciences. He is presently a professor for mathematics at Munich University of Applied Sciences.