Математика и информатика *Volume 62, Number 6, 2019* *Mathematics and Informatics*

Educational Matters
Научно-методически статии

DISCRETE MATHEMATICS AND PROGRAMMING – TEACHING AND LEARNING APPROACHES

Mariyana Raykova, Hristina Kostadinova, Stoyan Boev
New Bulgarian University – Sofia (Bulgaria)

Abstract. Discrete mathematics is one of the most important parts of the introduction to programming. Although known as the basis of writing good software and its usage is necessary, the students have many difficulties while gaining the competency to apply it in software projects. In order to make it a little bit easier for learners, an additional course Discrete Mathematics and Programming was conducted in the Informatics major in the New Bulgarian University. Its objectives, conduction, some exemplary tasks and homework are presented in this paper. The difficulties in the learning process and different approaches to reduce them are discussed.

Keywords: discrete mathematics; programming; algorithm; learning

1. Introduction

Discrete mathematics is one of the pillars of computer science. It gives future software developers fundamental knowledge for optimally solving standard tasks and helps them recognize if a given problem has a solution. The course Discrete Mathematics is included in all major/bachelor computer science programs in university. It pushes students to evolve critical and logical thinking, to find an appropriate solution to a certain problem and to create a mathematical model of a real-world situation to implement it in a computer program. The topics included and assessment approaches used in the course Discrete Mathematics vary in different universities (Discrete Mathematics for Computer Science, 2019; Gallier, 2009), (Mathematics for Computer Science, 2019). In some of them, students gain basic knowledge of all the terms and structures, but others give a deep understanding of most of the concepts and emphasize on the details.

To teach each mathematical course is always a big challenge. Discrete Mathematics is not an exception. The difficulty in the learning process raises several important questions that have no trivial answers:

1. How to motivate students to learn mathematics when they cannot see its application?
2. How to lead students through all the topics with increasing difficulty without losing them?
3. How to help students find a way to use in practice all new terminology, algorithms, and methods they have learned.

4. How to stimulate students' higher level of knowledge such as evaluation and creation (Bloom, 1956)?

When trying to answer these questions our team of lecturers in the department of informatics in the New Bulgarian University, created a new course Discrete Mathematics and Programming two years ago. The course is held in the fall semester in the first year and students in Informatics major are enrolled. As both discrete mathematics and programming are hard to learn, maybe because they are hard to teach, we will present an overview of the course and will try to summarize our experience by focusing on the answers to the previous several questions.

2. Course Objectives

In the course "Discrete Mathematics and Programming" students learn the main terms in discrete mathematics (Stein, 2010; Feil, 2002; Manev, 2012) and their usage in computer science by solving tasks and creating simple software programs in C++ programming language (Reingold, 1997; Eckel, 2000; Schildt, 1997). At the end of the course, learners can implement and modify popular algorithms of Number Theory, Coding Theory, Set Theory and Combinatorics. This course is the symbiosis between the two basic courses "Discrete Mathematics" and "Programming" and it helps students to find the easiest way of learning both of them. The stress of the course is to motivate learners to gain a deep understanding of the basic topics of computer science and competencies to implement different mathematical models in software projects.

3. Course Description and Conducting

The following topics are included in "Discrete Mathematics and Programming":

- Topic 1. Divisibility: Numeral Systems, Prime Numbers, Greatest Common Divisor (GCD) and Least Common Multiple (LCM)
- Topic 2. Modular Arithmetic
- Topic 3. Coding and Cryptography: Hash Functions, Affine Ciphers, ISBN Code
- Topic 4. Sets and Subsets: Counting, Generating, Coding
- Topic 5. Permutations: Counting, Generating, Coding
- Topic 6. Partition of positive integers and sets: Counting, Generating, Coding

The course is divided into two parts: theoretical and practical. It starts with the theoretical part where all the needed terms, rules and algorithms are discussed and trained "by hand". After the theoretical part, students are ready to implement all they have learned by writing software programs with escalating difficulty. Each part ends with an exam. The theoretical part ends with an online quiz with test items of different types. The exam at the end of the practical part consists of several tasks which solution is to create software programs by using the knowledge from the theoretical part.

The course is held for 2 consecutive years with first-year students. Its schedule has to be done according to the Programming course schedule because most of the competencies needed to write software programs are gained there. The whole set of terminology, syntactic phrases in C++ programming language and basic algorithms are supposed to be already taught in the first-year Programming course. When the course was held for the first time, three theoretical lectures were followed by 3 practical exercises. However, according to the students' opinion and the results of the grading, it seemed to be better to move the whole practical part after the theoretical one. During the second year, when the course was held, the whole theoretical part, which ended with a theoretical test was followed by the practical part, which ended with a practical exam.

4. Examples, Homework and Exams

We will present some sample materials from topic 4. Sets and Subsets. This topic covers the cardinality of set, operations with set and subsets through bitmasks and subsets generation/coding. As explained earlier each topic is presented first in theory and after that by using programs in C++. Only the practical part is included in this section. It starts with one trivial task and two different types of solutions. The new term in the topic is bitmask (Kenneth, 2006; Lovasz, 2006).

Definition: bitmask of a subset of a given ordered set is a sequence of 0-s and 1-s with length the cardinality of the set, where the 1-s corresponds to the included elements in the subset. For example **U={a,b,c,d,e}, A={a,c,d}**, the bitmask of **U** is **11111**, the bitmask of the empty set is **00000** and the bitmask of the set **A** is **10110**.

Task: generate all subsets of a given sequence of letters.

Two possible solutions of this task are presented during the course. The first one is an iterative solution and the second is a recursive one. Both solutions are explained and discussed with students, in order to facilitate them in implementing the algorithms (Eckel, 2000; Schildt, 1997).

Solution 1. Iterative approach, algorithm:

Step 1. Check if the given sequence of letters is a set. If it is not, remove the repeated letters (this implementation is presented in the previous topic in the course)

Step 2. Finding the cardinality of the set

Step 3. Creating dynamic two-dimensional array of characters with $\mathbf{2^n}$ rows and **n** columns, where **n** is the cardinality of the set

Step 4. Filling the array with all the binary representations of the numbers from **0** to $\mathbf{2^n-1}$

```
// convert number to its binary representation
int toBinary(int number, char* bin, unsigned sz) {
      int i, j=0, tmp;
      for (i = sz - 1; i >= 0; --i) {
            tmp = (number >> i) & 1;
            bin[j++] = (char)(tmp + 48);
      }
      bin[j] = ‚\0‘;
      return 0;
}

// write the binary representation into an array
int genBitmasks(char** bitmasks, unsigned cardinality) {
      unsigned rows = std::pow(2, cardinality);
      for (size_t i = 0; i < rows; i++)
      {
            toBinary(i, bitmasks[i], cardinality);
      }
      return 0;
}

// generate subset, by using the set and the bitmask
int* genSubSet(char* bitmask, int* set, unsigned
cardinality,
      int* subset, unsigned& subcardinality) {
      subcardinality = 0;
      for (size_t i = 0; i <cardinality; i++)
      {
            if (bitmask[i] == '1')subcardinality++;
      }
      if (subcardinality) {
            if (subset != nullptr) delete[] subset;
            subset = new int[subcardinality];
            for (size_t i = 0, j=0; i <cardinality; i++)
            {
                  if (bitmask[i] == '1') {
                        subset[j++] = set[i];
                  }
            }
      }
      return subset;
}
// print array
int printArray(int* arr, unsigned sz) {
      for (size_t i = 0; i < sz; i++)
      {
            std::cout << arr[i] << "\t";
      }
      std::cout << std::endl;
      return 0;
}

int main() {
      unsigned cardinality;
      cout << "please enter cardinality of universal set: ";
      cin >>cardinality;
      int * set = nullptr;
      set = new int[cardinality];
      unsigned rows = pow(2, cardinality);
      char** bitmasks = nullptr;
      bitmasks = new char*[rows];
      for (size_t i = 0; i < rows; i++)   {
            bitmasks[i] = new char[cardinality + 1];
      }
genBitmasks(bitmasks, cardinality);
int * subset = nullptr;
unsigned subcardinality;
for (size_t i = 0; i < rows; i++)   {
      subset = genSubSet(bitmasks[i], set, cardinality,
subset, subcardinality);
printArray(subset, subcardinality);
}
}
```

Listing 1. Generate iteratively all subsets of a given set.

The program consists of several functions, which execute the steps in the above algorithm (Listing 1). The first function converts a number into binary representation. The second function writes the binary representation of each number in the interval **[0; 2^n-1]** in an array. The third function generates the subset, by using the given set and the bitmask as parameters. One last function is created to easily print the generated subset. It is called **printArray()**.

Solution 2. Recursive approach:

```
// print set
void printSet(int * set, unsigned sz){
    for(unsigned i=0; i<sz; i++)
        cout << set[i] << "\t";
    cout << endl;
}

// generate subset recursively
void gen_sub_set(unsigned len, unsigned pos, int * set){
    if(len==pos){
        printSet(set, len);
        return;
    }
    set[pos] = 0;
    gen_sub_set(len, pos+1, set);
    set[pos] = 1;
    gen_sub_set(len, pos+1, set);

}

int main()
{
    int B[3] = {0};

    gen_sub_set(3, 0, B);
}
```

Listing 2. Generate recursively all subsets of a given set

The solution is shorter than the iterative one and it seems to be easier, because it looks more comprehensive. Unfortunately, students had a huge number of difficulties to understanding the second approach because it uses a recursive algorithm. Although recursion is covered in the programming course several weeks before this topic is presented, most of the students cannot understand the logic of the pre-recursive and post-recursive calls. The case is more complicated because of the second recursive call, the students are lost in the consequence of the function calls, and they do not know the actual content of the call stack.

Homework

After each lab in the course, the students have to do their individual homework that consist of several tasks. The solutions of all the tasks is important in order to help students achieve the necessary competency to create software projects and applying the discrete mathematics in programming. The next several tasks illustrate what is the difficulty of the homework and give some exemplary problems that can be used to practice the knowledge learned in the course (Kreher, 1998; Knuth, 2011; Reingold, 1977). These tasks are from Number systems and Sets and Subsets topics.

Task 1. Create C++ console application that converts a number from the base-10 (decimal) numeral system to another positional numeral system. The user has to enter two whole numbers **x** and **y** in the decimal numeral system, where **x** is the number and **y** is the base of the target numeral system, **y** belongs to the interval between 2 and 16 inclusive.

Task 2. Create C++ console application that finds all possible partitions of a given number **n** as a sum of numbers (**n** is entered by the user).

Hint: the representations of number 5 are:

5=5
5=4+1
5=3+2
5=3+1+1
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
5=3+2 and 5=2+3 are equal.

Task 3. Create C++ console application that checks if a given set B is equal to a given set A, by using bitmask. Hint: use subtraction of sets.

Task4. Create C++ console application that finds all subsets of a given universal set with **k** number of elements; **k** is in the interval between **0** and the cardinality of the set and is entered by the user.

Assessment and Final Grades

As explained earlier the assessment of the students consists of two separate exams: theoretical (after the theoretical part) and practical (after the practical part). Before the practical exam there is one lab for preparation for the practical exam. In this lab several tasks similar to these given on the exam are solved and variety of useful techniques are discussed. Each student can ask individual questions and the lecturer can pay attention on these questions in order to help students take the exam easily. In the next section one exemplary practical exam with two tasks is presented. The first task is given in order to evaluate if students have competency to apply their knowledge about sets and strings. The second task is about generating all subsets of a given set by using bitmasks. Similar algorithm is explained in two labs by presenting analogous tasks.

Sample Practical Exam

Task 1.

A Set of symbols and a string are given. Write a function that codes the longest sequence of symbols in the string, which belong to the set by replacing each symbol in this sequence with *. The coded string has to be shown on the console.

Input:
```
Set: {'o', 'n', 'i', 's', 'e'}
String: "This is one test string!"
```
Output:
```
"This is *** test string!" // the longest sequence of symbols
is "one" and it is coded with ***.
```
Task 2.

Write a function that generates all subsets of a given universal set, which must have elements on even positions. Subsets are given with a bitmask.
```
Input:
Universal set: {1, 2, 3, 4, 5}
Output:
1      0      1      0      1
1      0      1      1      1
1      1      1      0      1
1      1      1      1      1
```
Grades

The grades for both years when the course was conducted, can be seen in Figure 1 and Figure 2. The two diagrams show the results of the theoretical and practical parts of the exam and the final grades. The results of the theoretical part are higher than the practical ones and most of the students who failed the exam had problems with programming. The fact that the percentage of the students who didn't pass the programming part of the exam is extremely high has to be analyzed in detail to improve the learning process. Two important reasons need our attention:

The students cannot apply theoretical knowledge to use it to solve the practical tasks. It means that the higher level of knowledge such as analysis, evaluation, and the most important one - creation has to be trained with more examples and different activities (homework, assignments, etc.).

The learning content that is presented is too difficult for first semester students. The students must be more experienced in programming to use and modify the algorithms for subsets and permutation generation. The conclusion is that these algorithms have to be discussed at the end of the course and the students have to be already familiar with similar ones, learned in the programming course.

Another worrisome fact can be seen in both diagrams. There is a significant difference between the grades' distribution of theory and practice. The theory distribution is moved to the highest grades, but the practice is moved to the lowest grades. An additional possible reason for that, excluding the two, discussed earlier is that the activities used to assess theoretical knowledge and practical competency are quite different. The theory is graded by a quiz with multiple choice and fill in the blanks questions and the practical part is graded by an assignment with several tasks (exemplary exam is shown in the previous section) that has to be solved in an IDE, writing software programs.

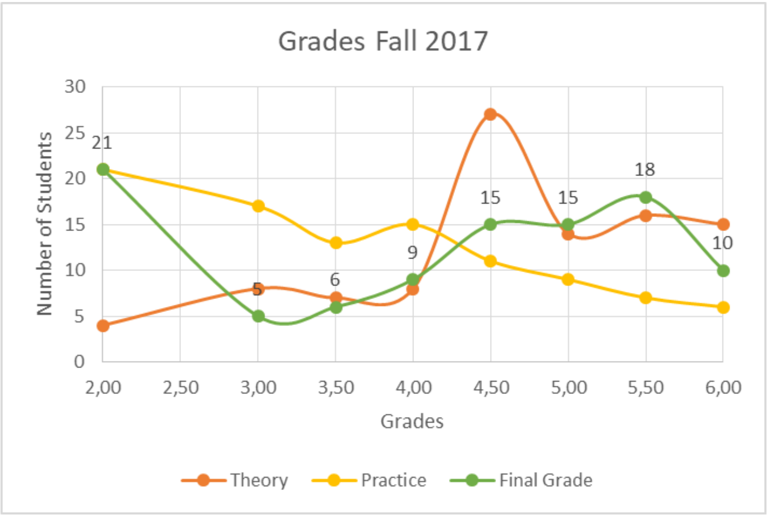

Figure 1. Course Final Grades – Fall 2017

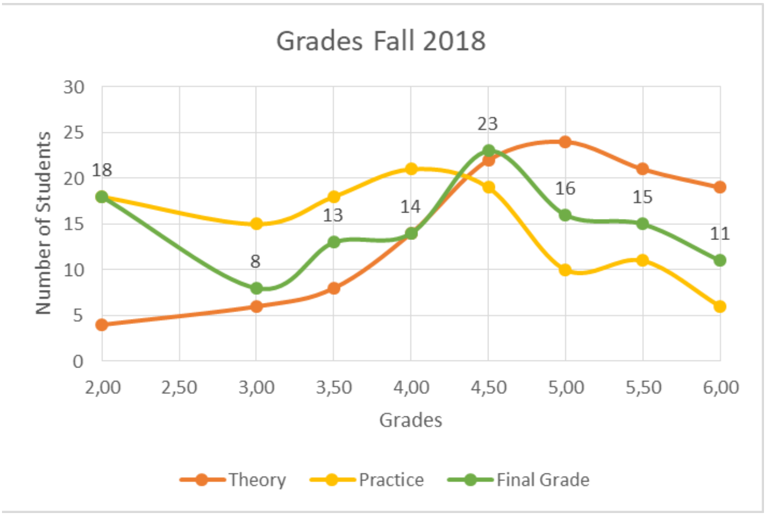

Figure 2. Course Final Grades – Fall 2018

Comparing the diagrams of the two years, we can recognize that although the percentage of the bad grades is high, the percentage of the students who failed the exam at the end of the second year is less than the one of the students who failed the first year. It is a good tendency that we hope will be held next year.

5. Learning Process Difficulties

As discussed in the previous section during the course conduction there were some difficulties in achieving the course learning objectives. We can improve the learning process by doing several things in the next year:

1. Recognize the most difficult topics in the course and try to pay more attention and give additional tasks on them. The most difficult topics are Sets and Subsets, Permutations, and partition of positive integers, and sets. The solution to this problem is not a trivial one, because the content of the other topics have to be reduced and a variety of new tasks in the difficult topics have to be included. A significant part of the course results improvement is the lecturer to find an appropriate way to force students not to give up and be patient to learn the difficult content. Some group tasks on the practical part can be given to increase motivation and collaborative learning.
2. Stimulate analytical and logical thinking and the most important competency – creation of students' solutions by applying the knowledge they have learned. One possible approach here is to use many discussions

where students are active participants. Another method can be to use one specific learning activity: the so-called peer assessment. Using this approach, students can revise their colleagues' solutions and try to improve them. Every revision has to be explained and the proposed solution is supposed to be better than the first one. The lecturer can assess both the first solutions and revisions. The main objective here is to try to engage the students in the process of creating and optimizing software programs.

Conclusion

During our efforts to make discrete mathematics easier to teach and learn, our team in the department of Informatics in the New Bulgarian University made one new course: Discrete Mathematics and Programming. It is included in the first semester of Computer science bachelor's degree. Course objectives, description and some sample homework and exams are presented in this paper. We emphasize on the difficulties in the learning process, analyze them and try to give some possible solutions of the most important ones. In the next years we are going to improve the assignments given in the course and the conduction of labs were the most difficult topics are given. In addition, we plan to make the course in two semesters in order to overcome some of the difficulties that we met – students' success is not satisfactory and the results they show are that they have covered knowledge in lower levels of Bloom's taxonomy of knowledge (Bloom, 1956).

NOTES

1. Discrete Mathematics for Computer Science. Last visited on 15 August 2019. https://lewis.seas.harvard.edu/pages/harvard-computer-science-20-discrete-mathematics-computer-science
2. Mathematics for Computer Science. Last visited on 15 August 2019. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-spring-2015/index.htm

REFERENCES

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H. & Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives, Handbook I*: *The Cognitive Domain*. New York: David McKay Co Inc.

Stein, C., Drysdale, R. & Bogart, K. (2010). *Discrete mathematics for Computer Scientists*, ISBN-13: 978-0132122719, ISBN-10: 0132122715, Addison-Wesley; 1 edition.

Knuth, D. (2011). *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms*, Part 1, Series: Art of Computer Programming, Addison-Wesley Professional; 1 edition, ISBN-10: 0201038048, ISBN-13: 978-0201038040.

Kreher, D. & Stinson, D. (1998). *Combinatorial Algorithms: Generation, Enumeration, and Search*, CRC Press; 1 edition, ISBN-10: 084933988X, ISBN-13: 978-0849339882.

Reingold, E., Nievergelt, J. & Deo, N. (1997). *Combinatorial Algorithms : Theory and Practice*, Prentice Hall, College Div, ISBN:013152447X.

Eckel B. (2000). *Thinking in C++,* 2nd edition, v.1, v2, Prentice Hall; 2nd edition, ISBN-10: 0139798099, ISBN-13: 978-0139798092.

Gallier, J. H. (2009). *Discrete Mathematics, Some Notes.* ScholarlyCommons, University of Pennsylvania.

Kenneth H. (2006) *Discrete Mathematics and Its Applications with MathZone, 6th edition*, McGraw-Hill Higher Education, 2006, ISBN 0073312711, 9780073312712.

Lovasz, L., Pelikan, J. & Vesztergombi K. (2006). *Discrete Mathematics: Elementary and Beyond*, Springer, ASIN: B00FB4BNG0.

Schildt, H. (1997). *Teach Yourself C 3rd Edition*, McGraw-Hill Osborne Media; 3 edition, ISBN-10: 0078823110, ISBN-13: 978-0078823114.

Feil,T. &Kroan, J. (2002). *Essential discrete mathematics for computer science*, Publisher: Prentice Hall, ISBN-10: 0130186619, ISBN-13: 978-0130186614.

Manev, K. (2012). *Introduction to Discrete Mathematics*, KLMN; V Ediiton, Sofia, ISBN: 9789545351365.

✉ **Dr. Mariyana Raykova, Assist. Prof.**
Dr. Hristina Kostadinova
Dr. Stoyan Boev
Department of Computer Science
New Bulgarian University
21, Montevideo Blvd.
1618 Sofia, Bulgaria
E-mail: mraykova@nbu.bg
E-mail: hkostadinova@nbu.bg
E-mail: stoyan@nbu.bg