Discrete Math with Programming: Better Together

Kirby McMaster
Computer Science Dept.
Weber State University
kmcmaster@weber.edu

Nicole Anderson
Computer Science Dept.
Weber State University
nanderson1@weber.edu

Brian Rague
Computer Science Dept.
Weber State University
brague@weber.edu

ABSTRACT

This paper proposes a Discrete Mathematics course that is integrated with programming. The course consists of a sequence of Math modules with coordinated programming projects. Advantages of this approach are presented, and a methodology for developing the course is shown. A sample list of Math modules and brief project descriptions are included.

Categories and Subject Descriptors:

G.2.0 [**Discrete Mathematics**]: General.

General Terms

Algorithms, Theory.

Keywords

Discrete Mathematics, object-oriented programming.

1. INTRODUCTION

In recent years, there has been increased discussion about the Discrete Mathematics (DM) course in Computer Science (CS) programs. The ACM Computing Curriculum 2001 document [19] includes DM (referred to as Discrete Structures) in the CS body of knowledge. This document specifies six core Discrete Structures topics, and recommends that DM topics should be integrated early into the introductory curriculum. One strategy suggested by the document is to "require computer science students to take courses in discrete mathematics concurrently with the introductory sequence." Another approach is to "integrate at least part of the material on discrete mathematics directly into the introductory computer science sequence." Typical implementations are expected to be a mixture of both approaches.

In this paper, we propose a DM course in which programming is an essential part of the course. By programming, we mean the entire software development process, including problem definition, design, coding, and testing. This DM course would not be in the CS1/CS2 introductory programming sequence, but would be taken after CS1. When DM includes programming, the programming exercises review concepts and refine problem-solving skills introduced to students in CS1. This is not a new idea, but it has been overlooked in recent presentations on teaching DM. The 1968 ACM curriculum guidelines [1] for a Discrete Structures course (Course B3) states: "This course provides the student with ... an opportunity to apply the programming techniques he has learned in Course B1 [Introduction to Computing]."

In our CS program, the DM course is scheduled immediately after CS1, and concurrent with CS2. The emphasis in our CS1 course has been structured programming, but is evolving into an "early objects" approach. The DM course is adapting by having the programming projects review structured programming while gradually introducing the use of classes and objects to represent DM models and algorithms.

Programming--transforming ideas into software--is the primary framework for learning in CS. During the DM course, students are able to use programming to help them grasp the mathematical concepts more easily. Conversely, the DM models and algorithms provide students with opportunities to improve their programming skills. CS students often perform better in a course when concepts are used to solve real problems. Many enjoy programming but have difficulty appreciating the value of DM until it is applied to a domain of interest.

2. REVIEW OF LITERATURE

Since 2000, over 30 papers on DM have been presented at ACM-sponsored professional meetings. Much of the attention in these papers has been on the following issues:

1. Is DM needed in CS?
2. Which department should teach DM?
3. How should DM be taught?

Reasons supporting the need for DM in CS range from the abstract to the practical. Ralston [16] states that mathematics is important because of its "effect on the mind." Bruce, et al [2] suggests that "specific topics in mathematics are not as important as having a high level of mathematical sophistication." Both authors agree that mathematical thinking can be helpful at all stages of software development. Tucker [20] feels that the CS 2001 curriculum guidelines are "math-phobic" in the sense that many theoretical and mathematical ideas that were prominent in the "CS of the past" are no longer part of the curriculum.

Another consideration is which department should teach DM? Ralston [16] suggests that the Math Department should teach the DM course because "CS/SE faculty members ... have quite enough to do teaching courses directly in the discipline without teaching *strictly mathematical* [italics added] courses". Decker and Ventura [3] argue that CS faculty must teach the course so that DM topics will be related to the material in other CS courses. Henderson, et al [8] describe how Math and CS educators are working together to develop DM requirements for the Mathematics curriculum. LeBlanc and Leibowitz [13] present a "model of collaboration" for a two-semester DM sequence, in which the first course is taught by CS faculty and the second course is taught by Math faculty. The general trend is for CS to control the DM course, even when it is taught by Math.

Most ideas on how to teach the DM course involve tying Math concepts to CS applications. Rarely is programming mentioned. Some strategies are simple, such as Suraweera's [18] approach that starts each lecture with a "story" about a real world problem. McGuffee [14] describes a more structured solution, the Discrete Mathematics Enhancement Project, which has developed learning exercises and Web links to interactive visualization modules. Henderson, Marion, and Epp [5,9] conducted two SIGCSE-sponsored workshops on "nifty examples" for DM. Page [15] analyzes a DM course based on materials from the Beseme Project, where two-thirds of the course focuses on logical reasoning about software. Krone and Feil [12] present a course that does integrate DM and programming, but their course is CS2 and not a separate DM course.

3. DISCRETE MATH INTEGRATED WITH PROGRAMMING

Before we demonstrate how to teach DM with programming, let us compare three general approaches to teaching the DM course: (1) DM as a traditional Math course, (3) DM with CS applications, and (3) DM integrated with programming. If DM is taught as a *traditional Math* course, the focus is on the elegance of mathematical logic, theorems, and proofs. The traditional Math version of the DM course includes topics such as sets, relations, functions, logic, methods of proof, counting, probability, graphs, and Boolean algebra. The normal classroom presentation style is lecture. Course assignments consist of homework problems from the textbook, including some involving proofs.

When the DM course is taught *with CS applications*, the focus is on how mathematical models and algorithms are used in CS. The course includes most of the topics listed above, but with less emphasis on theorems and proofs. CS-specific topics are often added, such as logic programming, digital logic, relational databases, models of computation, and data coding. Classroom activities include discussion of "nifty examples" and software demonstrations (e.g. a circuit design program). Course assignments include homework problems from the textbook, with an occasional Math function to code.

Teaching DM *integrated with programming* involves more than providing CS applications and having students write a few programs. The course must have a *dual focus*--a Math focus and a programming focus. For the *Math focus*, the Math topics must be organized into a logical sequence that allows each concept to be explained in terms of material presented earlier. For example, the standard textbook approach in DM courses is to offer *logic* as the initial topic, but we prefer to cover *functions* before logic so that logical expressions and truth tables can be defined as functions.

The *programming focus* in the DM course takes the form of programming projects that are synchronized with the Math topics. We recommend that instructors group the Math topics into a sequence of *modules*, and then design a programming project for each module. Projects should review language features covered in the CS1 course and gradually introduce new programming concepts as needed. Throughout the DM course, important programming issues should be discussed, such as computer representation of Math models, algorithm efficiency and accuracy, language feature limitations, and advantages of OOP.

Learning Math always requires motivation. We assign each programming project at the *start* of a Math module. This allows students to see which Math concepts they must learn before they write the programs. Course assignments include some homework problems from the textbook, but a substantial amount of classroom time is spent discussing the programming projects.

4. DESIGNING A SEQUENCE OF DISCRETE MATH MODULES

To develop a course that integrates DM with programming, the main task is to create a sequence of DM modules with coordinated programming projects. The process for designing the modules should answer the following questions:

1. Which *Math topics* should be included?
2. How should the topics be grouped into *modules*?
3. In what *order* should the modules be presented?

The *Math topics* to consider for the DM course should include the core DM topics from the ACM 2001 guidelines, along with other ACM recommended topics. Math requirements for later CS courses should be considered. Topics can also be found in the Table of Contents of DM textbooks, although the preferences of authors should not unduly influence the selection of topics.

To illustrate the design process, we form an initial list of Math topics using the course outlines for a two-semester DM sequence CS105 (Discrete Structures I) and CS106 (Discrete Structures II) presented in the ACM 2001 curriculum guidelines. Our list starts with the six core DM topics, followed by six additional topics:

1. Functions, Relations, and Sets
2. Basic Logic
3. Proof Techniques
4. Basics of Counting
5. Graphs and Trees
6. Discrete Probability.
7. Digital Logic and Digital Systems
8. Elementary Number Theory
9. Basic Algorithmic Analysis
10. Basic Computability
11. Complexity Classes P and NP
12. Matrices.

Each topic consists of a number of sub-topics and concepts, some of which are detailed in the CS105 and CS106 course descriptions. For example, Basic Logic includes propositional logic, truth tables, predicate logic, etc.

This initial list can then be modified based on the needs of other CS courses. In our CS program, instructors for three required courses have expressed a need for specific DM topics:

1. Database Systems - relations and predicate logic.
2. Computer Architecture - digital logic.
3. Theory of Computing - automata and grammars.

These topics are already included in the initial list. Predicate logic is a sub-topic of Basic Logic, and automata and grammars are sub-topics of Basic Computability.

How DM topics are grouped into *modules* depends on the criteria used as the basis for grouping. Our DM modules are based primarily on similarity of Math concepts, coupled with having each module require about three weeks. Less importance is placed on the types of CS applications for the Math models. From the above topics, we define the following (unordered) DM modules:

A1. *Functions* and *sets* from Topic 1; *properties of integers* from Elementary Number Theory.
A2. *Relations* from Topic 1; *matrices*; *graphs* and *trees*.
A3. *Propositional logic*, *predicate logic*, and *proof techniques* from Basic Logic; *digital logic*.
A4. *Counting rules* from Basics of Counting; *discrete probability*.
A5. *Automata* and *grammars* from Basic Computability.

The six core DM topics are represented in these modules, as are selected concepts from four of the remaining topics. *Algorithmic Analysis* and *Complexity Classes P and NP* are covered in later CS courses. We are able to include a fairly large number of DM topics because we are selective in the Math concepts we cover in each module, and our DM course is 4 semester-hours.

There remains the question of the *order* in which the DM modules should be presented. The ordering of modules, and topics within modules, should be based on both Math and programming considerations. When order is based on Math constraints, modules and topics should *logically* follow one another. If Topic Y depends on Topic X, then Topic X should be taught before Topic Y. For example, if *functions* are to be defined mathematically in terms of *sets*, then set concepts should be taught before functions.

When order is based on programming issues, then several criteria are possible. Two interrelated criteria we use are *programming model* (structured programming vs. OOP) and *datatype*. Since our DM course reviews CS1 concepts, the DM projects start with structured programming and gradually introduce OOP. Eventually, we expect all DM programs will feature classes and objects. Within this sequencing constraint, DM programs initially use simple datatypes (integer, character, logical, floating point) and progress toward more complex datatypes (array, string).

With these Math and programming criteria in mind, we order the DM modules and topics as follows:

B1. *Sets, functions, properties of integers*.
B2. *Propositional logic*, *predicate logic*, *proof techniques*, *digital logic*.
B3. *Counting rules, discrete probability*.
B4. *Relations, matrices, graphs, trees*.
B5. *Automata, grammars*.

Reasons that led to this particular module and topic ordering are presented in the following paragraphs. Other workable and consistent orderings are possible using other criteria.

In **module B1**, we introduce sets before functions, so that functions can be defined in terms of sets. Properties of integers are defined as integer functions (e.g. greatest common divisor). Programs for this module use integer datatypes (*int* and *long*) and structured programming.

In **module B2**, we present propositional logic in terms of logical variables and logical functions. In predicate logic, we focus on *predicates* as functions with a logical output value. Proof techniques are introduced here, and are used when appropriate in later modules. Digital logic is included in this module because it is based on propositional logic. Programs for this module use several datatypes to represent logical variables (characters F and T, integers 0 and 1, and Boolean *false* and *true*). The programs use structured programming, with an introduction to OOP.

In **module B3**, counting rules are presented before discrete probability and random variables, since some probability functions are based on counting. Programs for this module mix integer datatypes (usually *long*) and floating point datatypes (usually *double*). All counting and probability functions are combined into one “static” class (like a C-program library). Random variables are implemented as classes, with methods to generate random values.

In **module B4**, the emphasis is on relations, which are represented visually as directed graphs (digraphs) and are implemented in software as Boolean matrices. Undirected graphs are defined as symmetric digraphs. Trees are defined in terms of digraphs. Programs for this module define and use a Boolean matrix class. The class includes an integer *array* (0-1 values), and methods to perform matrix operations and to reveal properties of relations.

In **module B5**, we study automata and grammars as models for representing languages and computation. Programs for this module define a finite automaton class, which is used to instance objects that act as language recognizers. The class includes *strings* and integer arrays, plus methods to perform state transitions and to recognize strings.

After a module and topic sequence has been chosen, it is enlightening to compare how the topics are ordered in current DM textbooks. This exercise can provide one basis for selecting a textbook for the course. Table 1 displays, for the topics in our sequence, the chapter numbers in the latest editions of five well-known DM textbooks [6,7,10,11,17]. None of these textbooks support our topic sequence without some “chapter hopping.”

5. DISCRETE MATH PROGRAMMING PROJECTS

After the sequence of DM modules has been designed, the remaining step is to develop projects for each module that integrate Math and programming. We have previously indicated that our projects start with structured programming and gradually introduce OOP. Also, different datatypes are utilized during the project sequence. The goal of the programming projects is to help students learn DM by viewing models and algorithms in two ways--the Math view and the Computer view. The Math view looks at essential features and logical structure. The Computer view considers design trade-offs for different implementations. It is difficult to make good implementation decisions without a clear understanding of the Math models and algorithms. This is one of the main reasons for including programming with DM.

Table 1. Discrete Mathematics -- Textbook Chapters

Module	*Topic*	*Gersting*	*Grimaldi*	*Johnson-baugh*	*Kolman*	*Rosen*
1	Sets	3	3	2	1	1
	Functions	4	5	2	5	1
	Integer properties	2	4	5	1	2
2	Propositional logic	1	2	1	2	1
	Predicate logic	1	2	1	2	1
	Digital logic	7	15	11	6	10
3	Counting rules	3	1	6	3	4
	Discrete probability	3	3	6	3	5
4	Relations	4	7	3	4	7
	Boolean matrices	4	7	3	1	2
	Directed graphs	6	7	8	4	7
	Trees	5	12	9	7	9
5	Finite automata	8	6	12	10	11
	Grammars	8	--	12	10	11

Developing good programming projects for a DM course is difficult and time consuming. To assist instructors, we have prepared projects for each of our five DM modules. Each project can be completed comfortably in three weeks or less. A brief description of each project is presented below, along with the Math and programming concepts that apply to that project. Complete handouts for all projects are available upon request.

Project 1 - Integer Functions: Write code to implement several integer functions to examine the effect of using different datatypes and algorithms. Functions will involve integer division, primes, greatest common divisor, number bases, and sequences. Examples of recursive algorithms are included.

Math concepts: set, variable, function, domain, one-to-one, inverse, sequence, integer divisibility, factor, prime number, greatest common divisor, relatively prime, Euclidean algorithm, number bases. *Programming concepts*: datatype, variable, function, parameter, integer datatypes, integer overflow, integer division operators, selection, definite and indefinite iteration, recursion.

Project 2 – Propositional Logic, Digital Logic, and Predicate Logic: Write functions to calculate truth tables for propositional logic expressions involving AND, OR, and NOT operations. Write functions to calculate truth tables for digital circuits built using AND, OR, and NOT gates. Write predicates and rules to search for numbers having special properties (e.g. odd palindromes that are perfect squares). Also, create a class to represent ternary logic variables (3-valued logic).

Math concepts: proposition, logical variable, logical expression, truth table, predicate, gate, circuit, Karnaugh map. *Programming concepts*: character datatype, logical datatypes, predicate function, class, object, method, encapsulation.

Project 3 - Counting and Discrete Probability: Write functions for computing combinations and permutations. Write functions to calculate probability values for several discrete random variables (e.g. Binomial, Poisson). Include these functions as static methods in a library class. This class is not used to create objects, and all references to class methods use the class name. Apply the static methods to solve counting and probability problems. Also, create a class to implement Binomial random variables.

Math concepts: permutations, combinations, outcome, event, sample space, probability, random variable, probability distribution. *Programming concepts*: floating point datatypes, mixed-mode arithmetic, datatype conversion, static method.

Project 4 - Relations, Boolean Matrices, Digraphs, and Trees: Create a Boolean matrix class that includes methods to perform unary and binary operations, as well as closure methods (e.g. transitive closure). Write programs that use objects of this class to perform calculations on Boolean matrices and to examine properties of digraphs (e.g. cycles, weakly connected, tree).

Math concepts: Boolean matrix, Boolean matrix operations, relation, digraph, in-degree, out-degree, path, cycle, relation operations, properties of relations, closure of a relation, tree, root, connected, weakly-connected. *Programming concepts*: array, abstract datatype, Warshall's algorithm.

Project 5 - Finite Automata and Grammars: Write a FA class that encapsulates the features of a finite automaton. Include instance variables for the input alphabet, the state transition table, the set of accept states, and the current state. Include a state-transition method, and a method to determine if an input string is in the language recognized by the machine. Write programs that instance the FA class and check if various input strings are in the language of the FA. For each FA language, write a regular expression and a grammar that define the same language.

Math concepts: alphabet, language, state, accept state, finite automaton, regular expression, grammar. *Programming concepts*: string datatype, string functions, string operations.

6. SUMMARY AND CONCLUSIONS

Most recent papers on the Discrete Mathematics (DM) course have ignored programming. We propose a course that integrates DM with programming, so that programming projects facilitate learning Math, and Math applications improve software development skills.

The first stage in designing a DM course integrated with programming is to organize Math topics into a logical sequence of modules. We demonstrate this process, leading to a sequence of five DM modules.

A suitable programming project must then be developed for each module. We provide brief descriptions of projects for our five DM modules, along with the Math and programming concepts that apply to each project.

We have taught courses that integrate DM and programming for several years. Our DM course has evolved with changes in the CS1 course. The language in CS1 has changed from C to Java, and the focus has changed from structured programming to early objects. These changes allow a broader range of programming models to be included in the projects.

Course evaluations indicate that most students enjoy and benefit from having programming projects in the DM course. Much of this enthusiasm is probably due to the substantial portion of their grade that the projects represent. There is some evidence that the projects improve mathematical understanding, since performance on exams is positively correlated with project grades.

Just as some authors have argued that CS is becoming "math-phobic," we fear that CS is also becoming "programming-phobic" [4]. The heart of CS is creating software. In CS, what we do and how we learn is tied to software. Every CS course should reinforce software development methodologies. The DM course, in particular, benefits when programming is integrated into the course.

7. REFERENCES

[1] ACM Curriculum Committee On Computer Science, "CURRICULUM 68: Recommendations for Academic Programs in Computer Science." Communications of the ACM, Vol. 11, No. 3, March, 1968.

[2] Bruce, Kim B., et al, "Why Math?" Communications of the ACM, Vol. 46, No. 9, September 2003.

[3] Decker, Adrienne, and Phil Ventura, "We Claim this Class for Computer Science: A Non-Mathematician's Discrete Structures Course." Proceedings of SIGCSE 2004, Norfolk, VA.

[4] Denning, Peter, "Recentering Computer Science." Communications of the ACM, Vol. 48, No. 11, November, 2005.

[5] Epp, Susanna, et al, "More Nifty Examples in Discrete Mathematics." SIGCSE 2005 Workshop, St. Louis, MO.

[6] Gersting, Judith L., Mathematical Structures for Computer Science (5th ed). W. H. Freeman, 2003.

[7] Grimaldi, Ralph P., Discrete and Combinatorial Mathematics (5th ed). Addison-Wesley, 2004.

[8] Henderson, Peter B., et al, "Math Educators, Computer Science Educators: Working Together." Proceedings of SIGCSE 2003, Reno, NV.

[9] Henderson, Peter B., and Bill Marion, "Nifty Examples in Discrete Mathematics." SIGCSE 2004 Workshop, Norfolk, VA.

[10] Johnsonbaugh, Richard, Discrete Mathematics (6th ed). Prentice-Hall, 2005.

[11] Kolman, Bernard, et al, Discrete Mathematical Structures (5th ed). Prentice Hall, 2004.

[12] Krone, Joan, and Todd Feil, "Incorporating Mathematics Into the First Year CS Program: A New Approach to CS2." JCSC 17, 1 (October 2001).

[13] LeBlanc, Mark D., and Rochelle Leibowitz, "Discrete Partnership -- A Case for a Full Year of Discrete Math." Proceedings of SIGCSE 2006, Houston, TX.

[14] McGuffee, James W., "The Discrete Mathematics Enhancement Project." JCSC 17, 5 (April 2002)

[15] Page, Rex L., "Software is Discrete Mathematics." Proceedings of ICFP 2003, Uppsala, Sweden.

[16] Ralston, Anthony, "Do We Need ANY Mathematics in Computer Science Curricula?" SIGCSE Bulletin, Vol. 37, No. 2, June 2005.

[17] Rosen, Kenneth H, Discrete Mathematics and Its Applications (5th ed). McGraw-Hill, 2003.

[18] Suraweera, Francis, "Enhancing the Quality of Learning and Understanding of First-Year Mathematics for Computer Science Related Majors." SIGCSE Bulletin, Vol. 34, No. 4, December 2002.

[19] The Joint Task Force on Computing Curricula, "Computing Curriculum 2001: Computer Science." ACM/IEEE, 2001.

[20] Tucker, Allen B., "Our Curriculum Has Become Math-Phobic!" Proceedings of SIGCSE 2001, Charlotte, NC.