Computer Science in Elementary and Secondary Schools

Viera K. Proulx
College of Computer Science, Northeastern University
Boston MA, 02115 USA
tel. (617) 437-2462
Internet: vkp@ccs.northeastern.edu

Abstract

In this paper we examine the current computer science curriculum in elementary schools and identify its shortcomings. We then presents a new approach to teaching computer science at the elementary school level. We identify five main concepts that are at the heart of computer science, and show on examples from our experience how these concepts may be presented in the primary school classroom. The suggested approach encourages the students to spend most of their time thinking, inventing games, playing them, analyzing them, and explaining in words what is happening. The computer is used do visualize some of the algorithms, to execute simple programs, and to serve as the motivation for this approach. That means that a large portion of these activities can be carried out successfully with only a limited computer facilities. This opens up new opportunities for learning to the students in schools that lack financial resources.

Key Words: computer science curriculum, elementary education. problem solving.

Biography

Dr. Viera Krnanová Proulx, Associate Professor
College of Computer Science, Northeastern University
Boston MA, 02115 USA
tel. (617) 437-2462
fax (617) 437-5121
Internet: vkp@ccs.northeastern.edu

Viera Proulx has received a PhD in Computing Science from Columbia University in 1977. Since then she has been teaching computer science at Northeastern University and in 1982 helped to create the College of Computer Science. Her teaching interests cover introductory computer science courses, operating systems and computer organization, and advanced computer architectures. Her current research focuses on interactive visualizations and methodology for teaching introductory computer science. The software created with her colleagues, Richard Rasala, Harriet Fell, and Cynthia Brown has been selected as one of 101 Success Stories of Joe Wyatt Challenge in October 1991. Since 1990 she has been a member of Task Force of the ACM Pre-College Committee that is working on a Model High School Computer Science Curriculum.

Introduction

Today computers are used in many elementary school classrooms as everyday tools. Yet the students typically learn very little about what computers are made of, how do they work, and what makes them work. There is no satisfactory computer science curriculum for this age group. In this paper we outline five basic computer science topics that should be taught in elementary schools, and use examples from a fourth grade computer club to show a methodology for implementing such curriculum.

The five concepts are identified as follows: representation of data; learning about an algorithm as a set of directives; looking at computer as a machine that carries out the algorithm; teaching students to discover new algorithms and analyze them; and examining different algorithm based problem solving strategies. The emphasis is not on programming, but on learning how students can create their own algorithmic solution, on understanding the kind of reasoning the computer is designed for, and on learning about the limitations and power of computers. The approach used here for the elementary grades can be used as a basis for a high school computer science curriculum.

Computer Science Problem Solving Paradigm

Studying different subjects teaches students different ways of thinking - it exposes them to new problem solving strategies. For example in social sciences one often learns about using negotiations and compromise to achieve desired solution. Of course, there is supporting data, helping the problem solver to decide what are the possible outcomes for different compromises, but the point is, a major part of problem solving is based on evaluating different options to arrive at the **subjectively best solution**.

Mathematics and the sciences teach the students to solve the problems in an objective way. Here the **"rules of the game" are given** by different natural laws (expressed as equations in mathematics, physics, biology, or chemistry). Word problems, one of the typical kinds of exercises in both sciences and mathematics ask the student to identify the meaning of the known quantities, to find the formulas or equations that represents the situation described in the problem, and then to solve these

equations to find one or more unknown quantities. The problem solving strategy is reduced to repeating the thought process presented in the class. Some innovative approaches to teaching mathematics insist that the students participate in inventing the formulas and relationships [1] - an approach similar to the one proposed here.

In sciences students also conduct experiments - typically to confirm the natural laws that have been earlier presented in a lecture. Sometimes the experiments are set up so that the students can **deduct from the data collected** during the experiment **what the relevant law is**. This activity is more creative because student has to engage in some critical thinking to get the right result.

So how is computer science different from either sciences or the social sciences? The main difference is that the student in designing an algorithm (which is at heart of any kind of programming) is inventing the "rules of the game". To be successful, student must express the ideas clearly, without ambiguities, using a precise terminology (be it natural language or a programming language). There are typically several solutions and the student can explore the different options. Except for creative writing there is no other area of study in which student in the primary or secondary school has such a freedom to explore the different possibilities and engage in creative thinking as there is in computer science. As in mathematics, this will only happen if the teaching methodology supports this creativity.

There is more students learn from computer science. There is a great amount of discipline in this discipline. The program is not right until it runs under all initial conditions. This leads student to thinking about testing whether all possible inputs produce the desired results. Student learns to organize the work, to look for all possible "mishaps", to analyze carefully the given problem. Students compare different approaches and learn which one works the best for a given set of physical constraints.

Another important lesson is getting the basic understanding of how **computers** are built, **how do they work**, and **what are their limitations**. Every educated person should be aware of the shortcomings of computer systems, and the different possibilities of failure and misuse. Understanding the "insides" of the computer helps in understanding which problems are harder or easier to solve using a computer.

Teaching of Computer Science in Primary and Secondary Grades.

Given the argument in the previous section we may ask why isn't computer science a regular part of curriculum in all schools? Why are so many schools actually turning away from teaching computer science? Why isn't is embraced by all as the greatest way to introduce scientific type of creativity and discovery into the classroom? Why is it hardly ever taught in the elementary schools?

The most serious reason is that what many students study under the guise of computer science has very little to do with computer science. In many programming classes students first spend a great deal of time learning the syntax and semantics of a new language which they then use to represent algorithms. But the main job is the translation process - converting the algorithm from its English version to the representation in a programming language. Later, as new algorithms are introduced, the translation process grows in complexity - and in the level of frustration it brings with it. Towards the end of the year students may be given a simple project to work on, but by then their creativity has been squashed, they are used to hacking on the code until it somehow mysteriously comes out right - and they may run out of time.

Another reason is that often, after the "basic" algorithms have been covered, an inexperienced teacher runs out of interesting examples and problems suitable for the student's level of knowledge and programming ability. Students never learn to discover algorithms for themselves, or to analyze them, nor to compare different solutions.

If computer science is taught at the elementary school level at all, it is typically taught as a unit on LOGO programming. Students have an opportunity to discover algorithms for themselves, but are limited by the graphical world of the Turtle. Many students loose interest once they manage to teach the Turtle a few tricks - there are just so many pictures one wants to draw.

The question is: How can we teach the basic ideas computer science at the primary and secondary level, in such way that the students will not only engage in the critical thinking and problem solving typical for computer science, but also be able to carry the lessons learned into other subject areas? We outline the main topics and objectives that should be included in computer science curriculum in primary and

secondary schools. A Task Force of the Association of Computing Machinery is currently working on a proposal for a new high school curriculum in computer science in USA, that includes all of our suggestions [9]. Of course, the secondary school curriculum has more ambitious goals that those described here. In this paper we will describe the methodology that can be used in the elementary grades. Concrete examples are based on the author's experience with teaching a fourth grade computer club in the spring 1990. Similar methodology, based mainly on discrete mathematics related topics, has been used by M. Fellows [5].

The Topics and the Objectives

There are five main computer science concepts easily accessible to elementary school students. They complement each other and typically should not be taught in isolation. While trying to learn how to create a new algorithm, we may perform a simple analysis and evaluation. While describing an algorithm we can experiment with a number of different representations of the algorithm - in a language, as pictographs, as a 'secret' code, etc.

The first concept students should learn is the **representation of data**. Children at this age love secret codes. A Morse Code is an example of a useful code. The number of knots on a string used to represent different quantities, or Roman numerals are examples of representations of data that children find fascinating. Seeing the variety of data representation many children are familiar with, it should come as no surprise that they will be fascinated to learn that strings consisting of only two binary digits are needed to represent any kind of data. A simple number guessing game teaches them quickly how to represent all numbers between one and sixty-three in binary system. The magician has six cards, card number x contains all numbers whose representation as a binary number has a one in the x-th position (Figure 1). The first number on each card is the appropriate power of two. When the magician is told that the number is on the cards with 1, 4, and 16 in its first position, it is easy to guess that the number is 21. Children loved this game. They tried to write down all 63 different numbers - and discovered with delight that 101010 (representing number 42) is "opposite" of 010101

(representing 21). What a surprise that their sum is 63. How about other numbers that add up to 63? - The possibilities for exploration are wide open. The kids in computer club easily mastered binary addition and got some feel for representation of negative numbers as a two's complement (using the odometer analogy). Exercises of this kind teach the children to form *abstract representation* of the phenomena they are trying to understand. Once the foundation has been laid, this theme reappears again and again - and not only in the computer science setting.

The second concept is **algorithms as a set of directives** or operating instructions. I can hear the voices, "How in the world can you teach this to little children?". Well, if we look at what children like to do, they love algorithms. They love playing board games. The rules for Parcheesi constitute quite a complicated algorithm. Children learn it, can explain it to another person, and after playing the game a few times they begin to analyze it. They learn what moves are better than others, they devise their own strategies, test them against their opponent, and revise them as needed. There are many elementary school children that love playing chess. They may unknowingly fail to consider all the possible pitfalls of their next move, but luckily, their opponent typically plays the same way and the "blunders" even out [7].

The concept of algorithm should be introduced by first looking at a number of different algorithms they are already familiar with - rules for playing board games and card games, even sports, cooking recipes, directions for using appliances. They have seen other algorithms in their school work, e.g. algorithm for addition of two long numbers, or subtraction. There is even an algorithm for writing an essay or a book report - or for making a lunch. We can look at the different ways how the algorithms are represented. Some are described in plain narrative - and one has to make sure no steps are left out. (The instructions on a shampoo bottle say "Wet your hair, Lather, Rinse, Repeat" - but don't say what steps to repeat or how many times.) Others - like directions to your friend's house can be represented as a map. Instructions for small children or for foreigners are often described by a series of pictures (for example the emergency procedures in airplanes). This leads us to explaining the language that computer understands, the reason for it, and some simple examples.

The third concept is the notion of **computers as machines that carry out algorithms** - the concept of stored program, sequential execution of algorithm steps. Here exercises in programming a LOGO Turtle are quite appropriate, but many similar games can be devised. Function Machine [3] is another model for introducing programming at the elementary school level. Students learn that when they practice their arithmetic, they act as "human computers" - performing the addition and subtraction algorithms on the given set of inputs (the homework problems). A cook is a "computer" that performs algorithms described in a cookbook, using the ingredients as inputs, producing delicious edibles as outputs.

Next the students should get an extensive opportunity to **discover new algorithms** for themselves, describe them, **and analyze them**. Can young children really discover algorithms themselves? Take the simple card trick of guessing one of 21 cards lined up in three columns. After placing the column in which the mystery card occurs into the middle of the pack, dealing the cards again, and repeating this three times, the mystery card migrates to the position in the middle of the pack. Children discover what the algorithm is, even if they cannot verbalize their explanation why it works. Ask children to guess a number between one and one hundred. They quickly invent a somewhat convoluted binary search algorithm, apply it, and adjust it as necessary, until they come up with the result. Another example is a popular but silly game of Tic-Tac-Toe. By the time children are about eight years old, they know that if both opponents play the best strategy, neither can win. The children in our computer club described and analyzed the game of NIM on three rows and by recording the possible positions and transitions the class discovered the two crucial best moves needed to win the game. By the time we were done they were certain and convinced that you cannot win the game if your opponent starts, unless he makes a mistake. They can certainly describe a binary search algorithm - and will be fascinated to learn that one can guess one number out of 1,000,000 in only 20 guesses! They will go on for a long while repeating the guesses to verify this theory and will be thrilled each time it works. They learn, that the rules of games may favor one or the other player, or that neither of them can win. They find out that there is a formal precise method for describing the desired sequence of events.

Another algorithm our group had fun with was the Towers of Hanoi problem. They first tried to solve it themselves starting with five disks. They counted the number of moves and tried to do the best. They counted the moves for two, three, and four disks too. We then watched a dynamic visualization of the game on a computer (using the software developed for first year computer science curriculum [2]) . After seeing the optimal number of moves, and stopping the animation at the strategic spots children began to understand the recursive formulation of the solution. This lead us to trying to estimate the number of moves needed to move all 64 discs. It was quite fascinating to find out that if each move takes one second, the universe won't end for at least another 500 billion years. This number of moves is equal to the number of grains of rice promised in an ancient Indian folk tale. What are the children learning here? They are learning how important it is to describe the procedure leading to solution correctly, not forgetting any alternatives, they are learning that one can easily pose a problem and come up with an algorithm for which the length of time needed to carry out the solution is impossibly long - or where the solution is realized much faster than anticipated (binary search).

As the students are exposed to these concepts, they are also gradually learning a number of **algorithm based problem solving strategies**. *Divide and conquer* strategy appeared in the binary search game, *induction and counted loops* were used in some of the LOGO drawings, *decision and repetition that terminates on condition* appeared in the game of NIM, *recursion* solved the problem of Towers of Hanoi. A couple of other examples can be used to introduce *backtracking*, and *observing similarities* as a problem solving methods. Children were asked to try to solve at home the Eight Queens problem - placing eight queens on a chess board so that none of them can be taken by any other queen. We asked them to try this with boards of sizes 4, 5, 6, 7, and 8. Not all found a solution for eight queens, but all did it for the smaller boards. We then watched the animation that showed how a computer using backtracking arrived at a solution. Each time a new queen is picked the animation shades the squares that are endangered. The animation also points out if two solutions are symmetrical, and encourages the students to investigate the similarities between different solutions and different approaches to

solution. Another set of backtracking algorithms which children are very familiar with involves traversing mazes. Again, they can start at home by not only solving a few mazes, but also describing in words what their strategy was. It is not important that they master all these techniques, only that they become increasingly aware of them and begin to describe their logical reasoning.

Conclusion

We presented arguments for introducing computer science curriculum into elementary schools. Looking at the five different areas of this curriculum, it is clear that the students should spend most of their time thinking, inventing games, playing them, analyzing them, and explaining in words what is happening. The children also examine the resulting algorithms and evaluate them from a number of different perspectives. The computer is used do visualize some of the algorithms, to execute simple programs (LOGO, Function Machines), and serves as the motivation for this approach. The main objective is to teach children a new set of reasoning and problem solving skills -- namely the creating (rather than following) the rules of a game. The feasibility of the proposed curriculum is illustrated by the experience with a fourth grade computer club. The main emphasis is on thinking and learning how to solve problems, and can be done with minimal computer facilities. This opens up new opportunities for students in schools that lack financial resources for computers, a problem we should all try to address [6].

Acknowledgments

The author would like to thank Harriet Fell, Richard Rasala, Cindy Brown, Carol Wolf, Chuck Rice, Charles Bruen, Philip East, Susan Merritt, Gerry Segal, Darlene Grantham, Erich Neuwirth, Wally Feurzeig, and Mike Fellows for the numerous discussions that helped in shaping the ideas presented in this paper. Thanks are also due to the eleven children that loved to get up an hour earlier to come to the computer club -- Nadea, Jeff, Julie, Dan, Elenka, Peter, Beth, Chris, Josh, Kate, and Andrew.

References

(1) L. P. Bezenet, "The Teaching of Arithmetic I, II, III: The Story of An Experiment", The Journal of the national Educational Association, November (1935).

(2) C. Brown, H. J. Fell, V. K. Proulx, and R. Rasala, "Programming by experimentation and Example", in I. Tomek, ed., Computer Assisted Learning, Proceedings of the 4th International Conference ICCAL '92, Wolfville, Nova Scotia, June 1992 (Springer Verlag 1992), 136-147.

(3) R. Carter, W. Feurzeig, J. Richards, and N. Roberts, "Intelligent Tools for Mathematical Inquiry", in Proceedings of the 9th Annual National Education Computer Conference, Dallas TX, (1988).

(4) N. Casey, "The Whole Language Connection", Washington State Mathematics Council (1991).

(5) M. Fellows, "Computer SCIENCE and Mathematics in the Elementary Schools", to appear (1992).

(6) D. Tinsley, ed. Proceedings of the IFIP WC3.1 Working Conference "Impacts of Informatics on the Organization of Education", Santa Barbara, USA, August 1991, (Elsevier North Holland 1992).

(7) J. Johnson, "Observing the Massachusetts State Scholastic Chess Championship", Private Communication, 1984.

(8) B. Latour, "Visualization and Cognition: Thinking with Eyes and Hands", in Knowledge and Society: Studies in the Sociology of Culture Past and Present, 6 (1986) 1-40.

(9) S. Merritt, et. al. "ACM Model High School Computer Science Curriculum: Draft Report of the Task Force on High School Curriculum of the ACM Pre-College Committee", (1991).

(10) A. Zvonkin, "Mathematics for the Little Ones", J. Math, Behavior, (1992) to appear.

1	3	5	7	9	11	13
15	17	19	21	23	25	27
29	31	33	35	37	39	41
43	45	47	49	51	53	55
57	59	61	63			

2	3	6	7	10	11	14
15	18	19	22	23	26	27
30	31	34	35	38	39	42
43	46	47	50	51	54	55
58	59	62	63			

4	5	6	7	12	13	14
15	20	21	22	23	28	29
30	31	36	37	38	39	44
45	46	47	52	53	54	55
60	61	62	63			

8	9	10	11	12	13	14
15	24	25	26	27	28	29
30	31	40	41	42	43	44
45	46	47	56	57	58	59
60	61	62	63			

16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	48	49	50	51	52
53	54	55	56	57	58	59
60	61	62	63			

32	33	34	35	36	37	38
39	40	41	42	43	44	45
46	47	48	49	50	51	52
53	54	55	56	57	58	59
60	61	62	63			

Figure 1. Binary Card Game.