Computational Thinking: Some Food for Thought

MukarramUddin, Mateen Sultana

, MohdAfzal
Department of CSE
isl.hns.hod@gmail.com,sultanamateenisl@gmail.com
,mdafzal9islclg@gmail.com

ISL Engineering College.
International Airport Road, Bandlaguda, Chandrayangutta Hyderabad - 500005 Telangana, India.

ABSTRACT

In the same vein as reading, writing, and mathematics, computer science is now considered a foundational ability because to Jeannette Wing's advocacy for its inclusion in the curriculum. Mastery of computational thinking enables us to process information and tasks in a systematic and efficient manner, much as the communication and quantitative skills gained via mastery of the fundamentals of language arts and mathematics. While it's certainly a worthy objective, educating the masses to think computationally has some unique pedagogical challenges. Perhaps the most confusing question is whether or not teaching programming should be seen as a distinct discipline from teaching fundamental computer science. How much, if any, programming experience is necessary to be a competent CT user?

As such, we argue that efforts should be made to set the foundations of CT long before students encounter their first programming language in order to effectively increase involvement in computer science. We suggest that the relationship between programming and computer science is analogous to that between mathematics and proof creation or between English and literary analysis. Therefore, programming should not be a student's initial experience with CS, but rather the gateway into more advanced CS, according to analogical reasoning. We contend that, in the absence of programming, the primary goal of CT education should be to develop a shared vocabulary and set of symbols for annotating and describing computation and abstraction, suggesting information and execution, and providing notation around which mental models of processes can be built. Finally, we hypothesize that children who are regularly exposed to CT throughout their formative years would be better equipped for programming and the CS curriculum, and may even decide to study in CS for reasons other than professional prospects.

Keywords; Problem- solving using computation, language, and primary through secondary school instruction

INTRODUCTION

Despite the continued expansion of the IT sector, the computer science field has been struggling to recruit students since since the dot-com boom. We attribute students' lack of interest to uncertain job prospects, yet these kinds of arguments have been made, often even more forcefully, for years in other fields of study without having any discernible effect on enrolment. Neither the artistic and performing arts nor the social sciences and history have produced as many graduates as computer and information sciences in recent years, according to statistics from the National Center for Education Statistics1. This is hardly the stuff of ironclad job assurances. It's no surprise that the number of CS majors is far less than the number of students majoring in fields that are seen as more useful in the real world, like teaching or business. Although we have made great strides in explaining why computer science is more than "simply programming" over the years, the common idea that the two are interchangeable persists. This equation nevertheless gives a skewed and inaccurate representation of our field [6], which affects the number and makeup of our incoming students.

We think Jeannette Wing is on the right track when she advocates for the teaching of Computational Thinking (CT) [10] as a foundational ability on par with reading, writing, and arithmetic (the three R's). In a nutshell, Computational Thinking2 entails the following key points: 1) it is a method of problem solving and system design that draws on concepts fundamental to computer science; 2) it entails the creation and use of different levels of abstraction, to understand and solve problems more effectively; 3) it entails the ability to think algorithmically and to apply mathematical concepts to develop more efficient, fair, and secure solutions; and 4) it entails an understanding of the interplay between these factors. Contrary to popular belief, the goal of CT is not to train people to think like computers [10]. Rather, CT focuses on helping people acquire the wide range of skills needed to successfully apply computing to the solution of complex human issues [8].

1. PROGRAMMING: DESCRIBINGCOMPUTATIONALPROCESSES

There are landmark courses for students who want to go on to study English or mathematics at a more advanced level, which assist them transition from learning practical skills to learning the topics themselves. Literature courses in English prepare students for critical reading and theoretical debate. The first step towards more advanced mathematics is a course on how to read and write mathematical proofs. The ideas presented in these classes are groundbreaking. While having a high school diploma or GED indicates a certain level of literacy and numeracy, it does not guarantee familiarity with or ability in academic English and mathematics. We also see a divide between computational thinking as a talent and computer science as a discipline. This, therefore, is our thesis.

Like building proofs in mathematics or analyzing works of literature in English, programming is the backbone of computer science.

A program may be thought of as a concise, finite description of a large number of instances of computing, each of which represents a potentially endless process. Realistically, one has to know both the logic of the program and the constraints imposed on the logical artifact by the computer's hardware in order to comprehend how the program will run (e.g., an inte- ger is no longer an ordinary integer in the context of 16-bit representation). Students that are interested in advanced computer science should spend significant time actually writing programs to master these and related ideas. For only by focusing on the finer points of data representation

By learning about algorithm design, analysis, programming languages, and systems, students will be better equipped to take on advanced computer science topics. The standard, programming-first educational approach is suitable here.

However, programming shouldn't be required in order to declare programming literacy or as a prerequisite for teaching computational thinking. All students should have substantial expertise thinking computationally before beginning programming, much as math students come to proofs after 12 or more years of experience with fundamental arithmetic and English students come to literary analysis after an even longer time of reading and writing. Algebra, probability, and calculus, as well as English subjects like language arts, literature, and composition, aren't taught in elementary or secondary schools. Additionally absent are the plethora of electives that may be taken by non-majors in the English and Mathematics departments to improve their communication and analytical abilities. At the moment, students' first exposure to CT occurs in the same context where they first learn to code. This applies not just to the more advanced CS1/CS2 programs, but also to the plethora of CS0 programs aimed at providing various types of service. This instructional strategy is analogous to instructing students in elementary mathematics together with proof building, and to instructing students in primary reading and writing together with linguistics and discourse analysis. It's doable, however it may not be the best solution. When you don't have a firm grip on the processes you're trying to describe, it can't be simple to write descriptions in foreign formal languages. Our premise has the following consequence. Students should have substantial experience in computational thinking before enrolling in programming classes.

2. ALANGUAGEFORTHINKINGCOMPUTATIONALLY

Computational thinking has to be taught often and from a young age. In the absence of code, however, what does this entail? Understanding computational processes (and being able to conduct them manually) is more important than memorizing how they are implemented in any given programming language. Understanding the fundamentals of algorithms, such as the flow of control, is crucial. Learning to abstract and represent data, as well as analyze the features of different processes, is also crucial.

We talk about representing word problems and using algebraic techniques to derive simpler forms even in elementary school mathematics. However, we may also characterize the search space of feasible algebraic simplifications, which is caused by the beginning state, the end state, and the set of applicable operations. In this paper, we analyze the effectiveness of two different derivations and investigate the process of obtaining a derivation using a blind or heuristic search.

5Of course, we should also make an effort to reconsider the methods in which we transition students into programming, as was recently recommended [7].

employed to give semantic knowledge of computational processes by annotating and describing computation and abstraction, suggesting information and execution, and providing nota- tion [4, 8].

As Wing and others have shown, there are several openings at the secondary and basic education levels to introduce computational thinking [5]. Those in mathematics classes are the most noticeable. However, academics in the social sciences and the humanities are now learning what scientists in the biological and physical sciences have known for a long time: computer processes are ubiquitous. Find the right possibilities, write them down in the CTL, and work them into high school and middle school programs.

3. LEARNINGABOUTCOMPUTATIONALPROCESSES

Data representation and transformation concepts found in a CTL should be both familiar and fundamental. Concepts and notation offered in conjunction with the CTL should be acceptable for the grade level being taught. The remainder of this section provides a number of case studies illustrating how computational thinking may be included into elementary and secondary school courses. We emphasize the CTL vocabulary and notation that each uses to explain foundational CS ideas and concepts. Some of the examples we present are based on resources from the widely-used website edHelper.com, which also serves as a useful resource for information on the most recent changes in educational policy in the United States.

3.1: CTL Vocabulary

Around the third grade, when students are introduced to multi-step computations and elementary combinatorial problems, computational thinking is likely to make its first appearance in a student's academic career. At this point, introducing vocabulary in a planned manner may help bring about an understanding of computational procedures. Next, we look at a few examples that might help us understand how to use CTL terminology.

FIRST EXAMPLE (INTRODUCTION TO MULTIPLICATION). Multiplication is often taught to students in Grade 3 according to current curriculum standards. The idea that "multiplication is repeated addition" and the idea that "the outcome of multiplication is the same whether you write the smaller or the larger number first" are both rather popular.

By defining multiplication as successive additions, we can introduce the ideas of iteration and efficiency in computing. Each use of the plus sign may be thought of as an iteration, and while addition is commutative, the two modes of representation may have varying degrees of efficiency. These might be some good examples of exercises.

First, express each multiplication as a series of additions, followed by the product. Record the needed number of iterations as well.

Write the multiplication with the two numbers switched and see the difference in iterations. Which method is more productive?

Example

EXAMPLE2 (READING COMPREHENSION). As part of de-velopingtheirreadingcomprehensionskills,studentsinGrade3 are often given the task of putting a set of simple sentences into chronological order. Consider an example of such an ex-ercise: Given the four sentences

1: Idon’twantpizzaagainforalongtime.
2: Iateten pieces of pizza.
3: Later that night, I got sick.
4: Ifeltveryfull.

Which of the following sentence orderings is correct? a) 1,3,4,2 b) 4,3,2,1
c) 2,3,1,4 d) 3,1,4,2
e) 2,4,3,1

We may explain to students that each ordering of the four sentences is a *state*, and the five possibilities a) through e) make up the *search space* of the problem. To solve the prob-lem, we may verify each state individually, but we can al souse *divide-and-conquer* to *prune* in correct answers. The following are potentially suggestive home work questions.

1. Whatisthecorrectorderingbetween2and3?
2. Which of the states in the search space have 2 and 3 in the wrong order? Can the se answers be correct?
3. What are some other possible states not listed?

Later in middle school, when *permutation* is introduced, the search space concept may be revisited and broadened to those sentence orderings that result from applying the per-mute *operation* to some initial sentence ordering. Q

Here's a Third Sample (CHARTING INFORMATION). In elementary school, students are exposed to many different types of visual displays for data visualization, including pie charts, bar graphs, and tables. Most current activities on this subject focus on teaching students how to interpret each sort of display, although a comparison of computational benefits may also be explored. For the same set of data, a pie chart may label each slice as a percentage of the entire, and a bar graph might label each bar as an absolute number of the respective category. Consequently, the ratio between any two categories is more difficult to determine in a pie chart than in a bar graph, although the percentage of each category relative to the whole is "easier" to calculate in a pie chart. By having pupils count the number of mathematical operations needed to determine the percentage of each category in the whole, we may refine the idea of how simple or difficult something is. Q

expression	numbers switched	whichismoreefficient?

6	3×6	6×3	6×3needs3iterations, 3×6needs6,so6×3ismor eefficient.

assignment as an assembly line project, with students working on separate patterns or parts before coming together to create a whole. Improving throughput may be shown by keeping track of how much longer one procedure takes than the other while working on the same number of projects (along with some discussion and computation). By contrasting the finished output and the number of restarts due to mistakes, you can show how simplifying each step makes the process simpler and less prone to error for both humans and machines. Q

3.1 CTLNotation

Middle school is a good time to introduce children to CTL notation since it will be useful when they begin to confront more complicated calculations. In particular, a simple rewriting system to annotate state changes and basic tuple notation for organizing data and state representation might help to understand the computational elements of many issues.

Here's a 5th case in point: (FINDING Square ROOTS). Finding the square root of a number n without using a calculator is often recommended as a last resort, and a short Internet search confirms this. Each new estimate is obtained by dividing the current guess, g, by n and averaging that result with g. In computing, this procedure is known as estimate-divide-average (EDA). We may annotate the algorithm for determining the square root of 60, for instance, as follows, using a (bad) starting estimate of 2: 2 16 9.875.

⇒ 7.975 ⇒7.749 \s⇒ 7.746

When compared to a naïve calculation like a straightforward linear search (for instance, g.g + 0.1), we may explain that is an abstraction of the function g.(g/60 + g)/2. The idea of accuracy may be made a point of depending on the level of detail required. To provide just one example, linear search may produce unacceptable results or, worse, an endless loop.

One such possibility is to outline a binary search, whereby a range is expressed clearly and progressively decreased until its median is near enough to the solution. The contrasting calculation opens up a dialogue on representation and choice. Given that the next state is a unary function of the present state, a single number may perfectly characterize the situation in EDA. Each new state in a binary search is contingent on the existing upper and lower limits, hence some kind of pair representation is required. 7 Q

sentence → noun-phrase verb-phrase Noun-phrase: modifying noun | noun Verb Phrase: acting as a verb noun-phrase

The second guideline provides examples of alternative wordings for the noun-phrase. In conjunction with the previous rule, the grammar shows recursion. The method of drawing the diagrams may be shown either in the well-known parse tree format, or, to illustrate non-deterministic computational processes, as a derivation by recursive application of grammar rules.

Revisiting, Advancing, and Integrating

Previously presented concepts may be addressed at a more advanced level and with greater rigor as students go through their education.

7th INSTANCE: (EqUATIONAL REASONING). Equations are used in many different subjects, from physics and chemistry to business and biology, and students are progressively exposed to their application when they master fundamental algebra in middle school. Functional abstraction and procedural problem solving are essential CT abilities, and this is a great chance to demonstrate their use and strength.

The acceleration an of an object may be calculated from its velocity v and time t using the classic physics textbook equa- tion a = v. Essentially, the function's output type is acceleration a.

v, vJ, t, tJ.(vJ v)/(tJ t), which is utilised (i.e., used as a sub-procedure) in a number of other fundamental physical equations, including Newton's second rule of motion for calculating force, F = ma. In turn, Force F is an intrinsic part of many other physical equations. Such logical abstraction about functions, data types, and function composition is potent in its simplicity and clarity.

Q

Students gain a broader understanding of the importance of computational thinking (CT) skills in a wide range of fields when they are exposed to the subject matter in context, and teachers are better able to drive home the point that these skills are transferable and applicable in a variety of contexts when they integrate concepts.

SECOND EXAMPLE (INTERDISCIPLINARY PROJECTS).

Grammar may be better understood by drawing diagrams of sentences using the Reed-Kellog method or as trees in language arts programs. When learning how to draw a sentence diagram, the first lesson is usually learning how to locate the subject (phrase) and verb (words) in a sentence (phrase). Each sentence is broken down into its component parts, and then those parts are assigned their own linguistic categories. These kinds of drills are helpful for students to get used to a variety of notations, regardless of the representation system. In addition, the process of derivation and the representation of context-free grammars provide excellent venues for highlighting the efficacy of recursion and nondeterminism.

The formula is as follows: 7a, b.if (a + b)/2 > 60, (a, (a + b)/2); else ((a + b)/2, b).

Creating travel brochures is an interesting multidisciplinary endeavor. From as early as fifth grade, teachers might give their pupils a challenge that calls for knowledge of language arts, mathematics, and social studies (such as geography). The website of the Columbia Education Center is a great example. 8 An optimization version of the issue, for creating "excellent" tour packages over an area, may be discussed in high school (e.g., Eastern Europe). Scheduling restrictions such as travel time, budget, and interest in potential vacation spots all need to be considered. Instead of submitting just the completed

Furthermore, students may also be expected to display computational thinking by demonstrating several ways in which constraints may be stated and how (near) optimum solutions can be found in terms of an objective function. One such target function is to maximize the travel agency's profit while also satisfying the tourists' need for a certain level of enjoyment throughout their trip. Q

It may be beneficial to look for ways to include information processing into everyday classroom activities in addition to the explicit integration of CT and CTL in conventional disciplines.

9. EXAMPLE (GROUP PROJECTS). In scientific classes, group projects are commonplace. Each member of the group often has responsibility for one or two tasks exclusively, and interpersonal dynamics within the group are structured accordingly (e.g., data recording, report write-up). Introducing concepts like interface and encapsulation officially is best accomplished in contexts where two parties exchange data. When the relationships are reorganized in a new way, interesting problems emerge. For instance, if the project report has to be created in tandem (or asynchronously), then the team needs to talk about and implement notions like locking and message forwarding. Q

It is important to notice that computers are not mentioned directly in any of the examples in this section. Students are the computing agents throughout CT and CTL's infancy and early growth. In this context, students are seen as computers, and efforts are focused on improving their level of expertise and productivity. Naturally, software may be used to help with this, just as it is used to help teach elementary literacy and numeracy.
Our proposal differs from other major computer science education ef- forts such as [5] in that it does not focus on teaching students about how computation is implemented, but rather on incorporating the collective knowledge and lessons of computer science research into the discourse and development of all subjects that involve (information) processing. To this purpose, we need a group of computationally aware thinkers to do a fundamental reexamination of existing pre-college curriculum, to locate appropriate entry points for CT and CTL, and to develop appropriate instructional materials.

4. DISCUSSION

By the time they reach their senior year of high school, students will have had enough opportunity to think and speak in the CTL via practice and frequent interactions. Courses like "Great Theoretical Ideas in Computer Science" by Steven Rudich and colleagues at CMU9, which serve as good examples of beginning college courses, may assist students combine their experiences and get them ready for programming. Such programs may formally cover topics like convergence, efficiency, and limitations of computation without mentioning any particular computing agent. Domain-specific computational thinking courses at the university level are a possibility (e.g., bioinformatics, chemical informatics).
Students who continue their education in computer science by taking introductory programming courses will find that their greatest challenge is not in developing a computational mindset, but in mastering the subtleties of unfamiliar languages, learning to formally describe computations in those languages, and then learning how those descriptions are executed on a von Neumann machine. Even if students don't go on to study computer science, the skills they learn in this course will help them in many aspects of their lives and professions. Indeed, in this information era, it is critical to have a firm grasp of the applications and constraints of CT.

When it comes to the field of computer science, we believe that most students choose to major in mathematics, English, and the humanities not because of the plethora of job opportunities in these areas, but rather because of the intellectual interests that develop from long and repeated exposure to these disciplines. Students with similar levels of development in computational thinking are hypothesized to be more equipped for programming and the main curriculum, and they are also more inclined to pursue CS for intellectual purposes (cf. [3]). Students from more backgrounds will join our area as a result of being exposed to fundamentals of computer science since they will have a better idea of what CS (and informatics more generally) may be.
There is no question that it is difficult to effectively incorporate computational thinking into today's elementary and secondary school curriculum. The process must be progressive and evolutionary, and it will involve cooperation and coordination among various players in the education sector. We consider initiatives to improve the general public's knowledge of CT to be among the most interesting, difficult, and crucial next stages in the development of our field.

5. REFERENCES

[1] H. Abelson and G. J. Sussman[1]. Computer Programs: Their Structure and Interpretation, Second Edition, MIT Press, Cambridge, 1996.
[2] In [2] J. L. Bates and R. L. Constable. Programmatic proofs.
[3] It was published in 1985 in the ACM journal Transactions on Programming Language and Systems, volume 7(1), pages 113–136.
[4] Students who show promise in computer science don't always go in that direction, according to research [3] by L. Carter. Pages 27–31 in SIGCSE 2006, Houston.
[5] Asher Cohen and Benjamin Haberman [4]. The language of technology is computer science. SIGCSE advances, 39(4):65-69, 2007.
[6] Unplugged CS [5]. http://csunplugged.com.
[7] The work of P. J. Denning and A. McGettrick [6]. Information technology is coming back to the forefront. CACM 48(11):15-19, 2005.
[8] Author Reference: [7] M. Guzdial. Preparing the path for computational reasoning. CACM 51(8):25-27, 2008.
[9] Specifically, [8] S. Reges. Confusion about the meaning of "b:= (b = false)" The 2008 SIGCSE conference was held in Portland and was published on pages 21-25.
[10] Don Knuth: Part 2 of a Lifetime of Work Interrupted, edited by L. Shustek. CACM 51(8):31-35, 2008.
[11] 10 J. M. Wing, "Computational Thinking," CACM 49(3):33-35, 2006.