Data Structures and Algorithms in Pen-based Computing Environments

NAHIN

May 19, 2022

1 Introduction

We are entering a new era of teaching, learning and computing. With an abundance of useful information available on any subject, anywhere, anytime, and on any device, the challenge (faced by all us, the educators) is to create a teaching/learning environment that is mobile, smart, personable, and easy to use. Next generation's learning tools will require that we receive only the information we need at the right time. Future applications will also require natural interfaces that allow students (as well as instructors) to interact with any computing device with ease. The time for having "problem solving environments" in the classroom has emerged. A problem solving environment is an integrated computational system for solving problems in a specific application domain [1]. Such environments, in general, should enable the user to input and work on problems in a manner that is natural to the problem domain. The animation data structures and algorithms integrated with a natural pen-based interface is one of the most valuable environments in the classroom setting. Intuitively we, the computer science educators, believe that algorithm animation is helpful in teaching advanced computer science concepts. Students often study in either of the following two distinct modes according to their cognitive learning style. The first study mode is the initial learning of an algorithm and the associated data structure. In this mode students simply step through an algorithm to attain a preliminary grasp of the algorithm. This process of learning can be supported either by textual algorithm description or graphical algorithm animation. Some studies [2] have shown little improvement in student performance (mostly on analytical questions) due to the use of animations in teaching data structures and algorithms. Another study [3] has shown that simply viewing animation in a classroom setting did not significantly improve students' understanding. The second study mode is when students are active participants of the learning process. An example of this study mode is when students test themselves in preparation for an exam. In this mode a data structure visualization system must provide a high level of interactivity. It would be most helpful for students to have a system that will automatically verify the correctness of the student's manual algorithm execution. Although algorithm animation has been studied

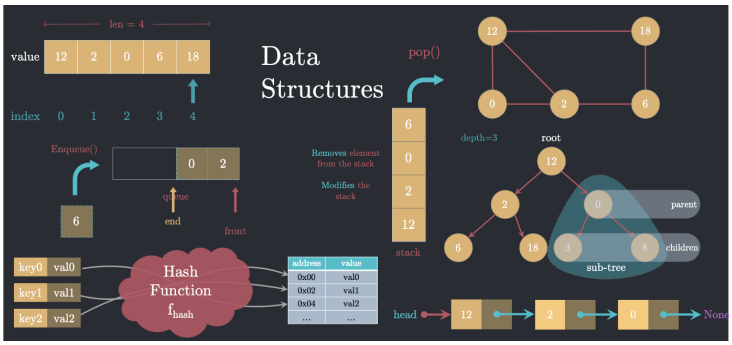

Figure 1: Data Structure and Algorithm.

for more than twenty years, the systems that have been proposed have not gained a wide acceptance in the classroom by students and their instructors. The reason is that most of the evaluations indicate that algorithm animation has no significant effect when students are just observers. We all know that students must be mentally engaged in order to learn. Therefore, we strongly believe that any new electronic educational technology integrated with a Tablet PC environment will have a fundamental influence on education in general. One of the principal aims of our study is to investigate the suitability of a Tablet PC environment in teaching data structures and algorithms. In this environment sketching is an integral part of computer science exploration. It allows the expression and exchange of ideas in a highly interactive atmosphere. Using the pen-based gesture interface, we will promote students' intuition for problem solving and algorithmic thinking.

2 SOLUTION EMPLOYED

Creating new interfaces is not a simple task; it has already been the subject of years of intensive research. Our work will be built upon research projects currently underway at MIT (see [4]) and CMU (see [5]). The primary goal of these projects is to develop a library of domainbased gesture recognition tools. Therefore, our project is not concerned with developing recognition tools but rather utilizing what is currently available. Our contribution to, and aim for, this is to automatically associate a geometrical drawing with the underlying data structure model. We restrict ourselves to simple geometrical drawings, such as a circle, a straight line, and a polygon. However, even in this trivial setting there is a lot of heuristics and uncertainties. As an example, consider the hierarchy of the following data structures: a graph, a tree, a binary tree, a search binary tree. Drawing each of these data structures requires only two graphical primitives: a circle and a line, so recognition does not represent a problem. Correct understanding and interpretation of the drawing is a whole different story. Domain-based interpretation is the cornerstone of our system. We restrict our-

selves to the following three most important topics in CS2: self-adjusted Binary Search Trees, Minimum Spanning Trees and Single Source Shortest Paths. Each of these topics includes one or more specific domains. The development of the interface can be broken down into four key parts: 1) stroke recognition and beautification, 2) the association of strokes to an underlying domain-dependent data structure, 3) the animation of algorithms, and 4) the verification of algorithms. The stroke recognition aspect comes into play when the user takes the stylus and draws a stroke on the Tablet PC. A series of points are digitized and then analyzed for feature points. The latter play a dominant role in shape perception by humans. If these characteristic points are identified properly, a shape can be represented in an efficient and compact way with accuracy sufficient for a given application. Various complicated corner detection algorithms have been developed (see [6].) Since drawings in our environment do not require high accuracy one can choose any of those algorithms. Once feature points have been detected, the next step is to classify the stroke and therefore to recognize the shape. For example, triangles can be detected by three feature points and the closed line segments between them. Furthermore, an important part of this recognition is the beautification of these strokes. In the example of binary search trees, when a student intends to draw a straight line, the system should snap the stroke to a straight line, beautifying the drawing. The same can be said of nodes; since most people cannot draw perfect circles, the system should snap elliptical strokes into perfect circles upon recognition with the condition that all nodes look the same (have the same radius). The next segment of this tool is the on-the-fly association of strokes with elements of the data structure. In the example of BSTs, each ellipse drawn will correspond to an existing node in the data structure and each edge between two nodes will represent a reference from the parent node to the child node. Consequently, each domain may require a set of heuristics in order to provide a simple, user-friendly interface. In the example of BSTs, with every pair of nodes that is connected by an edge, the node whose height on the screen is greater than the other will be designated the parent and the other node the child. The important part of this segment is that every edit to the canvas will result in an on-the-fly edit to the underlying data structure. It is important to mention here the equilibrium between a non-restrictive intuitive interface and theoretical correctness of the associated data structuremeaning that rules of the domain must not be violated. In the example of BSTs, no node may have more than two children, each node can have at most one parent, there are to be no cycles in any tree, and if there is data in the tree, the ordering property will be verified... etc. The last two major segments of this tool are the animation and verification of the algorithms as performed on hand-drawn data structures. The system will give students two different approaches for studying a particular algorithm. The first is when students draw a data structure and then watch a step-by-step algorithm animation. The second is when students draw a data structure and then graphically (manually) perform an algorithm execution having the system verify the correctness of each step performed. These two segments are of great pedagogical importance. This system does not dependent on any terse language, cultural

differences, nor "hand-wavy" explanations and arguments. Imagine the ideal situation when a PowerPoint lecture presentation seamlessly incorporates the algorithm animation tool. Such highly interactive multimedia environment can assist in demonstrating and exploring the algorithm concepts from various perspectives. Moreover, if every student had a Tablet PC, they can go home and practice on their own, the very same day, the things that were taught in lecture.

3 EVALUATION PLANS

A preliminary version of this new learning tool has been initially tested in 15-121 and 15-211 courses taught at Carnegie Mellon University in the summer of 2009. It is planned that systematic assessment of the learning outcomes will be conducted with the help of the Eberly Center for Teaching Excellence at CMU (see [7]). The Eberly Center is currently collecting data from a pilot group of students who are using Tablet PC's in an introductory computer science programming course. We will be using some of that technique to measure the usability aspect of the tablet PC and its effectiveness in improving student performance. We are hopeful to develop specific techniques to follow up with two groups of students. One group would take a traditional data structure course; the second group would be given Tablet PCs with the algorithm animation tool. The subject areas included in the evaluation would be binary search trees, AVL trees, splay trees and the minimum spanning trees. The visualization tool is intended to be used during a four week period. In addition, we plan to evaluate this tool according to students' cognitive learning styles based on the FelderSilverman style model [8, 9]. This model classifies students as: (1) sensitive or intuitive learners, (2) visual or verbal learners, (3) active or reflective learners, and (4) sequential or global learners. Depending on a particular style of learning, some students may (or may not) benefit from algorithm animation. It would be interesting to determine how algorithm animation on Tablet PCs influences different groups of students.Indeed, often the logical foundation of a proof argument seems to escape some of the students.

4 FUTURE PLANS: PEN-BASED PROOFS

Experience shows that many Computer Science students have great difficulties with the proofs methods encountered in, say, an advanced course on algorithms. Indeed, often the logical foundation of a proof argument seems to escape some of the students. We propose to transform students' experience with proofs by incorporating pen-based technology into introductory computer science courses. In particular, we consider formal proofs in Euclidean geometry. The cornerstone of this model is the concept of geometrical sketching dynamically combined with an underlying mathematical model. A completely natural way of drawing using a digital pen will generate a system of polynomial equations of several variables. The latter will be fed to a theorem prover (see [10, 11]), based on the Grőbner

bases technique, that will automatically establish inner properties of the model. Moreover, once a particular mathematical model is created and then checked for accuracy, it will serve as a basis for logical deduction of various geometrical statements that might follow. Lastly, a detailed step-by-step exposition of the proving process will be provided. While theorem provers have become surprisingly powerful over the last decade, they are still too cumbersome to use and too limited in their capabilities to have any impact as teaching tools. As a case in point, see the recent study by Freek Wiedijk [12] where the fifteen major systems available today are challenged to prove that the square-root of 2 is irrational. One way around this problem is to use a hybrid system that combines a pure theorem prover with a computer algebra system, like Mathematica and Maple. In such hybrid systems the theorem prover organizes and controls the logical part of the argument whereas the computer algebra system takes care of all algebraic manipulations. Application of this rule shortens some proofs tremendously, and also brings the argument much closer to the form that a human prover would employ. For our purposes such a simple pragmatic solution will suffice. To avoid interface issues we chose Analytica, a system that is implemented entirely within the computer algebra system Mathematica, see [13]. Sophisticated algorithms such as Grőbner bases and cylindrical algebra decomposition are readily available in this environment. Analytica is currently used (see [14]) as an auxiliary tool in a discrete mathematics course at CMU to motivate the strict formalization of reasoning.

5 Mathematics Equation

(a+b)^2 = $a^2 + b^2 + 2ab$

6 Table

SI no.	Subject	Marks
01	Data structure	A
02	Electronics	A+

7 REFERENCES

[1] From "Computer as Thinker/Doer: Problem-Solving Environments for Computational Science" by S. Gallopoulos, E. Houstis and J. Rice (IEEE Computational Science and Engineering, Summer 1994).

[2] J. Stasko, A. Badre, C. Lewis, Do Algorithm Animations Assist Learning? An Empirical Study and Analysis, Proceedings of the ITERCHI Conference Human Factors in Computing Systems, 1993, pp. 61— 66.

[3] A. Lawrence, A. Badre, J. Stasko, Empirical Evaluating the Use of Animations to Teach Algorithms, Proceedings of the 1994 IEEE Symposium on Visual Languages, 1994, pp. 48—54.

[4] MIT iCampus Magic Paper, http://icampus.mit.edu/MagicPaper/

[5] CMU Interactive Systems Laboratories http://www.is.cs.cmu.edu/js/

[6] R. Klette, A. Rosenfeld, Computational Geometry: Geometric Methods for Digital Image Analysis, Morgan Kaufmann, 2004.

[7] CMU's Eberly Center for Teaching Excellence http://www.cmu.edu/teaching/eberlycenter/

[8] R.M. Felder, L.K. Silverman, Learning Styles and Teaching Styles in Engineering Education, J. Engr. Education, 78, 1988, pp.674—681.

[9] R.M. Felder, R. Brent, Understanding Student Differences, J. Engr. Education, 94, 2005, pp. 57—72.

[10]L.C.Paulson, Isabelle – A generic theorem prover, Lecture Notes in Computer Science 828, SpringerVerlag, 1994

[11]S.-C. Chou, X.-S. Gao, J.-Z. Zhang, Machine proofs in geometry, Series of Applied Mathematics, vol. 6, World Scientific, Singapore, 1994.

[12]F. Wiedijk, "Comparing mathematical provers." In: A. Asperti, B. Buchberger J. Davenport (eds.), Mathematical Knowledge Management, Proceedings of MKM 2003, Springer LNCS 2594, 188-202, 2003.

[13]E. Clarke, M. Kohlhase, J. Ouaknine, K. Sutner, Analytica 2, Proceedings of the Calculemus Conference, Rome, 2003.

[14]K.Sutner, CDM: Teaching discrete mathematics to computer science majors, Journal on Educational Resources in Computing, 2005.