A Tool for Promoting Algorithm Development in Introductory CS Classes

Chrisila Pettrey, Suk Seo, Jungsoon Yoo, Sung Yoo, Zhijiang Dong
Computer Science Department
Middle Tennessee State University
United States
(cscbp, sseo, csyoojp, cssung, zdong)@mtsu.edu

Abstract: Beginning students in the Computer Science curriculum frequently do not appreciate the value of the design phase of any programming project. In addition, when asked to develop an algorithm for a given problem, they have difficulty distinguishing between a general solution and a completed program. As with other skills, what is needed is a way to practice applying the basic concepts of algorithm design. To that end, we have developed AlgoTutor (The Algorithm Tutor) – an online, graphical tool for practicing algorithm development. AlgoTutor is an interactive tool that provides automatic assessment of student designed algorithms. It supports both top-down and bottom-up design methodologies. In addition to the student algorithm design interface, there is an instructor interface that allows the teacher to create/edit problems and their solutions, manage student accounts, and analyze student grades as well as student activities.

1. Introduction

One of the most recommended Computer Science (CS) program outcomes is being able to analyze a given problem, identify the computing requirements, and develop an algorithm appropriate to its solution. However, it has been our experience that novice programmers do not see the value in designing an algorithm prior to writing code. Anecdotal evidence indicates that they believe developing an algorithm will increase the time necessary to do an assignment. Not surprisingly, their programming process tends to be a disorganized, trial-and-error method that produces ad-hoc solutions. As a result, many students struggle through the programming process, and end up spending a considerably large amount of time in debugging numerous logical or design errors which could have been prevented if they had taken time to carefully design the algorithm. On the other hand, if they are required to submit an algorithm for a lab assignment, they will turn in either a completed program or a nearly meaningless outline. This indicates that beginning programmers often do not have an understanding of the basic concepts of algorithm design. This lack of algorithm development skill is not due to a lack of stressing the importance in lectures. In the classroom, lectures are given on requirements analysis and algorithm development for nearly every example problem. At our university, we primarily use the top-down algorithm development method where students are taught to apply step-wise refinement to a top-level solution for a given problem, and then to convert the detailed design into a program. In addition to classroom examples, the students are specifically instructed to analyze the problem and develop an algorithm for each laboratory assignment before they start implementing the program.

After considering the disconnect between the importance placed on this topic by the faculty and the (lack of) ability of students to develop an algorithm, we came to the conclusion that students do not practice developing algorithms as much as they practice programming. This may be because they do not feel comfortable with the process and it is not as rewarding as writing code. But whatever their reason, what is needed is a tool that can be used to encourage students to practice developing algorithms. To that end, the tool should be web-based so students can practice at any time and any place. Because students might, therefore, be using the tool late at night and not have access to an instructor, the tool would need to provide automatic assessment. We also want to support both the top-down and the bottom-up design methodologies. And finally, the tool needs to have the ability for instructors to create/edit problems as well as analyze student performance.

In this paper we introduce AlgoTutor, an algorithm tutor that meets the aforementioned requirements and can be used to promote algorithm development skills in introductory CS classes. In the following section, we discuss other work related to our project. In Section 3, we give a description of the algorithm development process with AlgoTutor. Section 4 discusses the tutor system architecture including the database, the software, and the user interface. Section 5 describes results and lessons learned from the initial implementation of AlgoTutor. We conclude with future directions of the research.
2. Related Work

Many studies have been done on how to teach problem solving techniques in computer science classes (Allan & Kolesar 97, Bucci et al. 01, Evans 96, Gries 74, Arnow & Barshay 99). The top-down approach to problem solving was promoted in the 1970s (Dijkstra 97, Polya 73, Wirth 71) and has been recognized as one of the essential principles in software engineering. The current trend is to combine the top-down idea of starting with a high-level skeleton of a solution and refining each step repeatedly with the bottom-up approach of using pre-existing modules and then expanding until the system fulfills all the requirements for the project (Ginat 01, Ginat 02).

A number of tutoring systems (Nakabayashi et al. 97, Shah and Kumar 02, Yoo et al. 06) have been developed to help students in Computer Science learn problem solving techniques. ALVIS Live! (Hundhausen & Brown 07) is an environment that assists students in understanding the dynamic behavior of computer algorithms. This system allows the students to compose the algorithm in pseudo-code, evaluates each line of algorithm code, and provides immediate syntactic and semantic feedback by visualizing the effect.

Unlike ALVIS, there are other tools where algorithms are composed using flowcharts. Such systems include Iconic Programmer (Chen & Morris 05), FLINT (Ziegler & Crews 99), and RAPTOR (Carlisle et al. 05). In Iconic Programmer, programs are composed using three types of flowchart components: sequence, branching and looping. These programs can be traced one step at a time as well as being converted into a high-level programming language such as Java. The sequence components are limited to declaring a variable, assigning a value to a variable or outputting the value of a variable. FLINT is an environment that supports the process of implementation, testing, and debugging in addition to the algorithm development. The FLINT system forces students to start the problem solving process with design specification and algorithm development. The developed algorithms are evaluated through a flowchart interpreter and then can be traced step-by-step using the built-in debugger. RAPTOR offers additional flowchart components including “input” and “call.” The system allows developing code incrementally and it does not force top-down decomposition.

Alice (http://www.alice.org) is a tool to teach fundamental programming concepts by writing programs that manipulate 3D objects in a 3D world. In Alice's interactive interface, students put together a program by drag-and-dropping graphic tiles, which represent control structures and function calls. The programs are then visualized as animations.

The tutoring systems mentioned above include some useful aspects that help Computer Science students learn problem solving techniques or programming concepts, but none of the systems contains all of the features that are needed to train the students to practice algorithm development as we desire. AlgoTutor, the system that we are developing, is a web-base, interactive tool that allows students to practice both the top-down and bottom-up algorithm design methodologies by steering them through the process of designing a solution in a structured manner.

3. Algorithm Development with AlgoTutor

Technically, algorithm development with AlgoTutor begins with the instructor defining a problem. The system provides an instructor interface that allows teachers to create and edit problems and their associated solutions and operations. Once the instructor has defined a problem and stored an instructor solution in the system database, a student can start developing an algorithm to solve the problem. The automatic assessment tool uses the instructor’s solution to give the student feedback. Since alternate solutions are possible, AlgoTutor allows the instructor to list several solutions. If a student’s solution matches any of the instructor’s solutions, then it is considered acceptable. One useful attribute of AlgoTutor is that the algorithm development process for an instructor is nearly identical to that for a student. Therefore, we have combined the description for both instructors and students in this section. The next two subsections provide an explanation of some of the terminology used in the tutor. The final subsection contains a sample tutoring session.

3.1. Problems and Operations

In order to practice a problem solving skill, there must be underlying generic principles/concepts that a student works with in different situations. For example, if a student wants to be good at solving equations, then they learn generic principles such as you can add a number to both sides of an equation and you can multiply both sides of an
equation by a number. If the student wanted to solve the equation $3x + 4 = 7$, then they would need to use both of the generic concepts mentioned above.

Since designing an algorithm is a problem solving skill, there must be generic principles that are used to design algorithms. For example, a problem might be to add all the numbers from $m$ to $n$ and print the answer. To design an algorithm for this problem, we need several principles. Two such principles are *initialize a variable with a value* and *add a value to a variable*. But both these concepts can be used in designing many algorithms, i.e., solving many problems. In AlgoTutor, these generic concepts are called operations. Thus, on the highest level, in AlgoTutor, problems can be thought of as specific tasks to be solved whereas operations are statements/steps that can be used to solve problems. On a slightly more specific level, there are two types of operations – those that are primitive, and those that are complex. Primitive operations can be thought of as individual statements like the two operations mentioned above. Complex operations can be thought of as sub-problems such as *sort n numbers in ascending order*. By allowing sub-problems to be operations, AlgoTutor is able to enforce top-down design principles.

### 3.2. Operation Selection and Parameter Mapping

When a student wants to solve an equation, they have in their minds a list of possible operations they could perform. Admittedly, half the battle is finding out what operation to use. But at some point, in order to actually solve the equation they have to associate the variables in the generic operation with the specific problem. For example, consider the generic operation *add a number to both sides of an equation*. If the equation were $x + 2 = 4$, then the appropriate mapping of the variable *a number* in this instance would be $-2$, producing the specific operation *add $-2$ to both sides of the equation*. This illustrates the fact that there are really two parts to successful problem solving – operation selection and parameter mapping. We have found that these two phases apply to algorithm development as well.

In the past, when we asked our students to write an algorithm, one of four things usually happens. Some students just turn in the program. Of those who actually try to write an algorithm, some just repeat the problem description, some are completely wrong, and some would be correct if the right variables were used. (Fig. 1) shows three representative incorrect student algorithms. The algorithm in Sample 1 is a mere restatement of the problem itself. The algorithm in Sample 2 is completely incorrect, showing the student could not choose the correct operations and missing the iteration step necessary to search the array. The algorithm in Sample 3 shows the student basically knew which operations to use, but could not appropriately map the parameters.

Adhering to the two part problem solving process, in AlgoTutor, we have divided the algorithm design process into two phases. The first phase involves knowing which operations to use and what order to use them in. We call this the operation selection phase. In the second phase the student must map the variables in the generic operations to problem specific values. We call this the parameter mapping phase. These two phases are described in more detail in the next subsection.

<table>
<thead>
<tr>
<th>Sample 1:</th>
<th>Sort an array of $n$ numbers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use a bubble or selection sort to sort the array from lowest to highest</td>
<td></td>
</tr>
<tr>
<td>2. return and print the array to the screen</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample 2:</th>
<th>Find the smallest value in an array:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. if first &lt; second</td>
<td></td>
</tr>
<tr>
<td>lowest is the smallest of first and third</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>lowest is the largest of second and third</td>
<td></td>
</tr>
<tr>
<td>2. return the lowest value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample 3:</th>
<th>Find an average of 10 numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. while value &lt; 10</td>
<td></td>
</tr>
<tr>
<td>value1 + value2</td>
<td></td>
</tr>
<tr>
<td>value ++</td>
<td></td>
</tr>
<tr>
<td>value total / 10 = AverageValue</td>
<td></td>
</tr>
<tr>
<td>2. return Average Value</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1:** Samples of incorrect student algorithms.
3.3. A Sample Tutor Session

When a student logs onto AlgoTutor to design an algorithm for a particular problem, the first screen they see describes the top-level problem to be solved along with a link to the page that will allow them to begin developing the algorithm. (Fig. 2) shows a sample initial screen. The problem description is in the left column of the table while the link to start algorithm development is the word solve in the right hand column.

If the solving link is clicked, the student will be directed to a page similar to the one shown in (Fig. 3). The problem description is at the top of the screen for reference. Along the right hand side of the screen are operations that will be used in designing the algorithm for this particular problem. One difference between the instructor screen and the student screen is that the instructor has access to all operations in the database whereas the student only has access to those that pertain to the given problem. The operations are divided into top-level operations and lower level operations. The lower level operations are ones that are used inside of a top-level operation. For example, \textit{add next value to sum} would be part of a loop that is defined by \textit{while counter is less than n add it to sum}. 

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|}
\hline
\textbf{description} & \textbf{operations correct} & \textbf{parameters correct} \\
\hline
Level 1: & not solved & solve \\
\textit{Assuming two integer values m and n have been read (given).} & not solved & solve \\
1. Uses the while loop to find the sum of integers \textit{m} through \textit{n} inclusive. & not solved & solve \\
2. Displays the resulting sum. & not solved & solve \\
\hline
\textbf{Examples:} & & \\
Summation of integers 1 to 5 is 15 (1+2+3+4+5). & & \\
Summation of integers 1 to 10 is 55. & & \\
Summation of integers 16 to 13 is 46. & & \\
\hline
\end{tabular}
\end{table}

\textbf{Figure 2:} The initial problem presentation screen.

\textbf{Figure 3:} The initial operation selection phase screen. Operations which should be used to solve the given problem are shown on the right.
In order to develop an algorithm, a student (or instructor) drags the operations from the right side to the left side. Operations that are on the left can be reordered by dragging. There are buttons at the bottom of the screen that enable the student to check the current ordering of operations, read help pages, or send questions to the instructor. Clicking the Check my ordering button will provide immediate feedback (see (Fig. 4)). (Fig. 5) shows the operation selection phase after the student has dragged all operations to the left. Notice that the first two operations, initialize counter and initialize sum, actually have the same underlying operation assign a value to a variable. The underlying operation is the only one in the AlgoTutor database, but an instructor is allowed to add comments to an operation to make it problem specific, and those comments are what the student will see.

![Image of AlgoTutor operation selection phase](image)

**Figure 4:** The student solution is graded each time the Check my ordering or Check my mapping button is pressed.

<table>
<thead>
<tr>
<th>Problem Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assuming two integer values m and n have been read (given).</td>
</tr>
<tr>
<td>1. Uses the while loop to find the sum of integers m through n inclusive.</td>
</tr>
<tr>
<td>2. Displays the resulting sum.</td>
</tr>
</tbody>
</table>

**Figure 5:** The operation selection phase screen after dragging operations from right to left to create an algorithm that solves the given problem.

Also of note in (Fig. 5) is the presence of several question marks in the algorithm. These question marks represent the variables in the generic operation that need to be mapped to problem specific values. When a student believes the algorithm is correct (usually by getting a 100 after clicking the Check my ordering button), then they can click the Ordering done, parameter mapping next button. At this point, the student’s solution is checked against the instructor’s standard solution. If the two do not match, then the solution is compared to the alternate instructor solutions. An instructor solution may have a par-block (parallel execution block) to group operations which are not order dependent. This block is not shown to students in the operation selection phase, and this feature allows a single instructor solution to cover many alternative correct solutions. Once the operation selection phase is completed and the student grade is recorded, the standard solution is loaded for the parameter mapping phase.
Algorithm Tutor
Welcome, Chrisla Petey. Logout Feedback

Problems

In the parameter mapping phase, students replace the question marks with problem specific values. (Fig. 6) shows an initial screen of a parameter mapping phase. Students can replace each question mark either by clicking on the question mark and selecting an appropriate value from a pop-up menu or by dragging a variable tile from the right side to the question mark. Vertical bars around variables or literals indicate that the instructor locked those variables so the student has no choice. This eliminates the need for some alternative solutions and helps students figure out appropriate parameter mappings. For example, in (Fig. 6), it is correct to initialize x before you initialize y.
sum, but the locking forces the student to do it in the order given while at the same time providing some hints for a correct mapping.

After the student finishes the parameter mapping phase, AlgoTutor will display the web page illustrated in (Fig. 2), but the progress bar would be filled in, the grades for each phase would be displayed, and the solve link would be replaced by View Solution Done. The View Solution link enables the students to view the correct solution. While the example in this section was a simple example, AlgoTutor does support complex problems containing multiple levels of sub-problems. For a complex problem, the development steps for abstract level 1 would be like those shown in (Fig. 2) – (Fig. 6). After finishing the parameter mapping phase of level 1, AlgoTutor would display a screen like that shown in (Fig. 7), and students can then follow the links to develop the algorithms for the sub-problems. Solving a sub-problem prior to solving its parent problem is a form of the bottom-up approach as is reusing previously solved sub-problems. AlgoTutor has the ability to allow students to link to sub-problems without linking to the parent problem.

4. System Architecture

All the software used in AlgoTutor, such as MySQL, Apache, PHP, is 100% open source software. The AlgoTutor production server was built on the Linux operating system. However, using WampServer, one can reproduce the production server on the Windows operating system. The system architecture of AlgoTutor is shown in (Fig. 8). The server maintains the database, the tutor and the management software. The database stores and manages information regarding users, problems, operations, solutions, student activities, and feedbacks in a MySQL relational database. The course management software is for teachers to create/edit student accounts and to view and monitor student scores and activities. The problem/solution management software is for creating/editing problems, solutions, operations, and help questions and concepts.

On the client side, the algorithm composer is a web-based graphical user interface that supports drag-and-drop operations. It was developed using the PHP scripting language to generate XHTML web pages. Cascading Style Sheet (CSS) was used to format the web page colors, fonts, layout and other aspects of web page presentation. JavaScript was used to enable drag-and-drop user interaction and asynchronous HTTP requests via Ajax. Ajax programming gives a lot of power and flexibility for developing browser-based user-interfaces that provide user experiences that closely resemble desktop applications. The use of Ajax technology provides an interactive and responsive browser user interface. Access to the lab is controlled by the Apache and PHP authentication system and each student is identified by using a user id and a password. To limit and control the access to other user’s solutions and the management software, we created three levels of access rights: instructor, assistant, and student access. Student identification is maintained by using cookies during each session.

![Figure 8: The system architecture of AlgoTutor.](image-url)
5. Experiments

We conducted experiments to analyze the effectiveness of enforcing algorithm development prior to programming. We used AlgoTutor in some of the programming assignments in CS-II courses during the Fall 2008 semester, and compared the performance on the programming assignments for the students who used AlgoTutor with the performance on the same assignments for students who did not use AlgoTutor. Our experimental results show that using AlgoTutor prior to coding improves the performance of the programming assignments. (The detailed results of the experiments are reported in [Yoo et. al. 09].)

We also conducted an exit survey to obtain student feedback concerning using the AlgoTutor system in Fall 2008. (Fig. 9) summarizes the results. Overall, students showed strong interest in the system and provided positive feedback. Students responded to all of the survey questions as either strongly agree or agree with one exception. All questions received positive responses. The question that received the least favorable score was “Operation descriptions were clear.” One possible explanation is that you have to know how to solve a problem to build an algorithm. If you don’t know the solution, operations that are used in a teacher’s solution may not make much sense. This problem can be mitigated by helping a student understand the problem better by asking some warm-up questions or having the student review the related concepts for the problem.

<table>
<thead>
<tr>
<th>Survey Rating: 1.(Strongly agree), 2.(Agree), 3.(Neutral), 4.(Disagree), 5.(Strongly disagree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The algorithm tutor helped you understand the process of...</td>
</tr>
<tr>
<td>Using the algorithm tutor helped you spend less time coding.</td>
</tr>
<tr>
<td>You have a good understanding of the algorithm you...</td>
</tr>
<tr>
<td>The algorithm tutor helped you organize your thoughts and...</td>
</tr>
<tr>
<td>Showing system generated parameters was helpful.</td>
</tr>
<tr>
<td>The process of choosing parameters helped you figure out...</td>
</tr>
<tr>
<td>It was easy to define and use parameters in operations.</td>
</tr>
<tr>
<td>“Check ordering” button was helpful.</td>
</tr>
<tr>
<td>Selecting operations from the provided list helped you...</td>
</tr>
<tr>
<td>It was easy to find operations you want to use.</td>
</tr>
<tr>
<td>Operation descriptions were clear.</td>
</tr>
<tr>
<td>Not worrying about C++ syntax helped you concentrate on...</td>
</tr>
<tr>
<td>The layout of the Algorithm Tutor was easy to use.</td>
</tr>
<tr>
<td>The labels used on the buttons in the Algorithm Tutor were...</td>
</tr>
</tbody>
</table>

Figure 9: The result of the exit survey.

6. Conclusion and Future Work

AlgoTutor is a pedagogical tool that trains introductory Computer Science students to start the program solving process with algorithm design using pseudo code. The system has been developed to address the undesirable attitude that developing algorithms prior to writing code is not necessary. Using AlgoTutor, students can practice algorithm development skills in a web-based, interactive environment. The system provides online feedback for their algorithm so that students can work at their own pace at a convenient time. The system also provides a convenient graphical user interface for teachers so that they can prepare exercises, assess student performance, and get insight for the improvement of student learning. Our survey shows that students feel that AlgoTutor helped them organize their thoughts and develop algorithms. They also feel that Algotutor helped them reduce the coding time. In the
upcoming semesters we plan to run more experiments with new features such as providing warm-up questions and reviewing related concepts.

References


Acknowledgements

This research was supported by the NSF CCLI grant DUE-0737257.