Determining What the Student Understands - Assessment in an Unscaffolded Environment

C. W. Liew(✉) and H. Nguyen

Lafayette College, Easton, PA 18042, USA
{liewc,nguyenha}@lafayette.edu

Abstract. Assessment of skills and process knowledge is difficult and quite different from assessing knowledge of content. Many assessment systems use either multiple choice questions or other frameworks that provide a significant amount of scaffolding and this can influence the results. One reason for this is that they are easy to administer and the answers can be automatically graded. This paper describes an assessment tool that does not provide scaffolding (and therefore hints) and yet is able to automatically grade the free form answers through the use of domain knowledge heuristics. The tool has been developed for a tutoring system in the domain of red black trees (a data structure in computer science) and has been evaluated on three semesters of students in a computer science course.

Keywords: Assessment · Computer science
Unscaffolded environments

1 Introduction

One of the keys to developing effective tutoring systems is the ability to determine what a students knows or understands both before and after a tutoring session. Without this capability, we would be unable to either develop effective tutoring strategies or to evaluate the efficacy of any teaching or tutoring approach. However assessing a student's knowledge, skills or conceptual understanding can be a difficult task. It's not enough to determine what they can or cannot do but if we are to help students improve, we have to be able to narrow down to the specific issues that are a problem and also the contexts in which they occur.

This paper describes a tool that accurately assesses student knowledge and skills without needing any scaffolding that would bias the assessment. The tool consists of two modules that provide a non-scaffolded test taking environment and corrects the student answers to the tests. The grading module identifies (1) correct and incorrect answers, (2) where the first error occurs in incorrect answers and (3) the type of error in the majority of the cases. The tool and the

R. Nkambou et al. (Eds.): ITS 2018, LNCS 10858, pp. 339–344, 2018.
https://doi.org/10.1007/978-3-319-91464-0_37

associated tutoring system have been implemented for the domain of balanced binary trees, an important data structure in computer science. The tool has been successfully used to assess three semesters of students taking the data structures class at our institution.

2 Problem Domain: Balanced Binary Trees - Red Black Trees

Binary trees are a fundamental data structure in computer science and are commonly taught in the second computer science course (CS2). Balanced binary trees are a natural extension of binary trees and provide good performance for sorting and searching regardless of how the input data is arranged. Red black trees are one particular type of balanced binary trees - others include AVL [4] trees and 2–3 trees [1]. With data structures like these, students have to know how to program them (use them) also have to understand the underlying concepts underlying the operations. The basic insertion and deletion operations are much more complex when compared to those of a standard binary tree.

A red black tree is a self balancing binary search tree that has the following properties [5]:

1. The nodes of the tree are colored either red or black.
2. The root of the tree is always black.
3. A red node cannot have any red children.
4. Every path from the root to a null link contains the same number of black nodes.

The top-down algorithms [5] to insert or delete a value from a red-black tree starts at the root and, at every iteration, moves down to the next node, which is a child of the current node. At each node, it applies one or more transformation rules so that when the actual insertion (or deletion) is performed no subsequent actions are needed to maintain the tree's properties. Other types of balanced trees also use a similar approach. In our work we used red-black tree as an exemplar to evaluate our ideas and implementations, but they should be applicable to balanced trees in general. The transformation rules for insertion are called *color flip*, *single rotation*, *double rotation*, *simple insertion*, and *color root black*. There are more rules for deletion and they are called *color flip*, *single rotation*, *double rotation*, *simple deletion*, *switch value*, *drop*, *drop&rotate* and *color root black*.

3 Assessment and Related Work

Many tutoring systems have designed tests that accept answers in a restricted input language (e.g., numerics only). This leaves the problem with a large solution space that renders guessing ineffective. Previous work on developing assessment tools in unscaffolded environments used domain knowledge to infer structure and reasoning. For example, the PHYSICS-TUTOR system [2] used knowledge of basic physics to heuristically determine the intent of students when

they are solving mechanics problems in introductory physics courses. They used domain knowledge, algebraic knowledge and heuristics to accurately determine correctness and errors even if there were missing equations or factors and the answers contained numbers instead of variables.

In the red black tree domain students are assessed not for their knowledge of content but their skill in applying the insertion and deletion algorithms both of which require that a student knows how to apply the transformations and can recognize when they are applicable. In addition, students can either combine several base (canonical) steps into a larger one or can change the order in which transformations are applied (typically with a *color flip*). Multiple choice questions are not a good mechanism for assessing skills in this domain. The ideal assessment tool would handle these issues and provide an environment similar to a test where students are provided with a blank sheet of paper and free response questions. The questions would test the students understanding and the tool would allow the students to make the same mistakes that they would make on the paper test. The tool would also be able to grade the exam and determine each student's problem areas.

4 The Tutoring and Assessment Interfaces

The system has 3 sections - the pre-test, the tutor, and the post-test. In the test sections, a typical insertion (deletion) problem for red-black trees involves inserting a sequence of numbers to a starting tree (or deleting from it). Students have to show the state of the tree after every insertion/deletion; they are also encouraged to show any intermediate states (the trees that are created along the path to the solution). To this end, the test interface displays a "blank" binary tree canvas of 31 empty nodes. The student can click on any node to specify its value and color - submitting a tree is therefore equivalent to placing all of its nodes in the appropriate position in the tree canvas; nodes that are left empty are assumed to be null black nodes. The interface is designed to look like a sheet of paper with blanks to fill in - in this way, we ensure that the system does not provide any hints or clues as to what the desired answer would be.

In the tutoring section, students perform the same task of inserting to (or deleting from) a starting tree. However, a node-by-node modification of the current tree is not required; instead, students only need to select a node and the transformation to apply at that node from a drop-down list. The tutoring system follows the *granularity* approach and requires the student to always select a node and then a transformation. If the student's selection is incorrect, the system will immediately display an error message and provide feedback. If the selections are correct, the system will apply the chosen transformation and update the trees - the student does not have to manually update the tree.

5 The Assessment Tool

Assessment is more than just determining whether an answer is correct or incorrect. A good assessment tool will also produce information about the type of

error and the context to help the instructor or tutoring system determine how to best help a student resolve the error in her knowledge. The answers generated by a student when taking a test (pre or post) in our system are input to the grading tool which analyzes them and reports whether the student's answer(s) were correct or incorrect and additionally where the first error occurred, the type of error and the context in which the error was made. This information is used to help us build a model of the student's knowledge and the work on building and using the student model is reported in a companion paper [3].

There are three different types of possible errors: (i) incorrect node selection, (ii) incorrect transformation selection, and (iii) incorrect transformation application. Furthermore, some tree transformations can be applied in more than one context; for example, color flip at the root node is performed differently from color flip at a non-root node. The student might know how to correctly select and apply a transformation in one context but not in a different context. For each error, we would therefore also like to know the context in which it occurred.

The grading algorithm uses the constraints imposed on binary search trees (ordering and relationship of nodes) and its knowledge of red black trees to construct the canonical solution sequence. The key assumption is that the algorithm has a sufficient set of transformations (both primitive and those that are combinations of primitives) to recognize all transformations that a student might make. The algorithm assumes that a student will never combine more transformations than the system has. Thus the algorithm does not have to consider any intermediate steps that the student makes. If there is a step that the system does not recognize, then the step is skipped and the next step is checked to see if it is the result of some step or combination of steps. The result of the comparison steps is a sequence of transformations that the algorithm has identified. This sequence is compared to the sequence from the solution to try and find a match taking into account that a different ordering of transformations can also lead to a solution. This approach is sufficient to determine whether an answer is correct in all the cases that we have evaluated. If the answer is incorrect, the algorithm then proceeds (starting from the first step) to compare the answer and the solution step by step. Heuristics are used to modify the canonical solution to take into account macro-steps and reordered steps. These heuristics are sufficient to analyze most of the student answers that we have seen.

6 Data and Analysis

We evaluated the grading tool on three semesters of student data - Fall 2016, Spring 2017 and Fall 2017 - from a data structures class at our instituion. There was a total of 105 students from the three semesters.

The concepts underlying insertion were taught in lecture for one week and in the next week the students were evaluated over two days. On the first day, the students were given a pre-test for thirty minutes followed by a session with the tutoring system. Two days later, they were given the post-test which was a duplicate of the pre-test. The number of *unrecognized* errors is approximately

10% in the pre tests and 4% in the post tests. We analyzed the *unrecognized* errors by hand and found that in most cases we (the human graders) were also unable to determine the error type. Many of these errors were generated by students who were "gaming" the system to finish the tests faster by making random selections and then submitting the random answer. The data shows that the most common error was committed when selecting the appropriate transformation to apply. This error is caused by the student either (1) selecting the wrong node as the current node or (2) not correctly recognizing the preconditions for each transformation. The data provided in the student answers is insufficient for us to disambiguate between those two errors.

Table 1 shows a breakdown of the data by the type of transformation. The largest number of errors were made in applying the *color flip* transformation. This is one of the simpler transformations in that the colors of three nodes (current node, two children) are flipped. Most of the errors occurred due to the students not recognizing the applicable preconditions and selecting the transformation. Note that the grading tool only grades up until the first error so that if students were to make subsequent errors those errors would not be detected. This explains why the errors in *single rotation* and *double rotation* do not appear to decrease significantly (in some cases they increase) between the pre and post test. The *color flip* transformation frequently leads to a succeeding rotation and if a student does not apply the *color flip* they will not be tested on the second transformation. The tutoring helps the students recognize when to apply the *color flip* (decrease in errors between pre and post tests) and that leads them to an error in the subsequent rotation.

Table 1. Errors for each type of insertion operation

Semester	Color flip	Single rot	Double rot	Insertion	Recolor root	Unrecog
Pre F16	42	19	13	12	2	7
Post F16	15	16	17	6	1	2
Pre S17	67	25	19	30	3	13
Post S17	26	24	19	7	2	6
Pre F17	20	11	5	2	8	4
Post F17	12	6	9	3	1	4

Deletion operations are more complex than insertion operations and what makes it more difficult is that many of the operations share the same name as the insertion operations even though the preconditions and semantics are different. Just like for insertion, the most common error was committed when selecting the appropriate transformation to apply. The grading tool was able to correctly determine the type of error in approximately 90% of the pre and post test errors. Table 2 shows a breakdown of the data by the type of transformation. The largest number of errors were made in applying the *drop&rotate* transformation. This is a case that is poorly explained and illustrated in the textbook.

Table 2. Errors for each type of deletion operation

Semester	Color flip	Single rot	Double rot	Deletion	Recolor root	Switch value	Drop & rotate	Unrecog
Pre F16	1	15	19	4	6	11	50	9
Post F16	2	10	10	3	12	3	31	6
Pre S17	3	37	36	2	3	20	72	14
Post S17	1	18	22	2	10	5	63	13
Pre F17	0	14	14	0	1	19	35	11
Post F17	3	5	4	1	12	3	33	3

7 Conclusion

This paper has described an a tool for assessing student skills on red black tree (specialization of binary search tree) insertion and deletion algorithms. The advantage of the tool is that it provides a scaffolding-free (thus realistic) evaluation environment while still being able to accurately determine the correctness of student answers. In addition, it can classify 90% of the first error found in each student's answer. The tool uses strong domain knowledge and heuristics to determine the likely type of error in each case and has been evaluated on three semesters of student data from a data structures class. A companion paper shows how the analysis from the assessment tool can be used to construct an effective Bayesian student model.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Boston (1974)
2. Liew, C., Shapiro, J.A., Smith, D.: Determining the dimensions of variables in physics algebraic equations. Int. J. Artif. Intell. Tools **14**(1&2), 25–42 (2005)
3. Nguyen, H., Liew, C.W.: Building student models in a non-scaffolded testing environment. In: Proceedings of International Conference on Intelligent Tutoring Systems (2018)
4. Sedgewick, R.: Algorithms. Addison Wesley, Boston (1983)
5. Weiss, M.A.: Data Structures & Problem Solving Using Java, 3rd edn. Pearson Education Inc., London (2011)