A New Approach To Teaching Red Black Trees

Franceska Xhakaj
Department of Computer Science
Lafayette College
Easton, Pennsylvania 18042
xhakajf@lafayette.edu

Chun Wai Liew
Department of Computer Science
Lafayette College
Easton, Pennsylvania 18042
liewc@lafayette.edu

ABSTRACT

Red black trees are considered an important data structure and students can find it to be challenging and difficult to learn. Many approaches to teaching red black trees have been tried but not very successfully. This paper describes our new approach, the granularity approach, to teaching the top-down insertion algorithm for red black trees. Past approaches have focused on teaching the mechanics of applying the rules (color flip, single rotation and double rotation). The new approach is based on the hypothesis that students have more difficulty selecting the correct rule than in applying a selected rule. Our approach focuses on helping students learn how to correctly select the rules to be applied We supplement classroom lectures with an intelligent tutoring system that incorporates our approach. The approach and the tutoring system were used and evaluated in a small data structures class in the fall semester of 2014. The early results indicate that our approach and tutoring system are effective at helping students learn the top-down insertion algorithm.

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: Computer science education; E.1 [**Data Structures**]

Keywords

Balanced trees, red black trees

1. INTRODUCTION

In a computer science curriculum data structures provides students with the knowledge and skills that are fundamental in later parts of the curriculum, such as in databases, networks, architecture, algorithms, etc [3]. Data structures are part of the programming fundamentals and core topics in a computer science curriculum [1]. Red black trees (and all balanced trees) are an important data structure with many applications. Learning the red black tree algorithms can be quite challenging and difficult for the students for many reasons. Many approaches to teaching red black trees have been tried but they have not been very successful. These approaches have focused on teaching students the mechanics of applying the transformations (color flip, single rotation, double rotation) associated with red black trees. This paper describes a new approach for teaching the **top-down insertion algorithm** for constructing and maintaining red black trees. The new approach focuses on the recognition of the applicability of the transformations based on the hypothesis that students have more difficulty determining when to apply a transformation rather than how to apply it. We tested and evaluated our approach in a data structures class in the fall semester of 2014. The early experimental results validate our hypothesis and indicate that the approach is successful in helping students learn the top-down insertion algorithm for red black trees.

ITiCSE'15, July 6–8, 2015, Vilnius, Lithuania.

DOI: http://dx.doi.org/10.1145/2729094.2742624.

2. RED BLACK TREES

A red black tree is a self balancing binary search tree data structure, that has the following properties [9]:

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.

A red black tree must display all of the properties listed above. In addition, every operation performed on a red black tree such as insertion or deletion, should preserve these properties resulting in a changed, but still correct red black tree.

The top-down insertion algorithm described in [9] starts at the root of the tree and in a single iterative pass, modifies the tree by applying one or more of the insertion rules described below and eventually adds a new item to the tree. A new item is always inserted in a leaf position in the tree, and it is colored red.

Just as with a binary search tree, during the top down traversal the algorithm selects the next node to examine based on a comparison of values at the current node and the value to be inserted. In addition, the algorithm uses the *current node* reference to keep track of its position in the tree and also to determine which rules (see description below) are applicable at the current node. The *current node* is marked with an X in each of the rules shown in Figure 1.

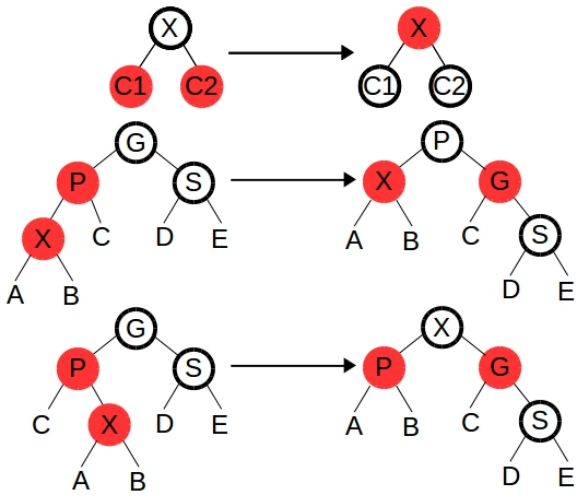

Figure 1: Top-Down insertion rules (top to bottom): color flip, single rotation, double rotation. X is the current node and relative to X, C1, C2, P, S, and G are the left child, right child, parent, sibling and grandparent. A, B, C, D and E are other nodes which may or may not be present.

Figure 1 shows the rules for top-down insertion in red black trees [9]. Starting from the top we have *color flip*, *single rotation*, and *double rotation*. In addition, we consider as a fourth rule *simple insertion*, insertion of a leaf node into the tree. There is also a fifth rule *color root black*, which is applied at the end of every insertion operation to ensure that the root of the tree is always black, as specified by the second red black tree property. The first rule (color flip) is to minimize the transformations arising from inserting a new leaf node that is colored red. The second (single rotation) and third rules (double rotation) are used to correct any violations (two red nodes in sequence) that arise from either applying rule 1 or by inserting a new node (simple insertion). The rules together ensure that only a single traversal from the root of the tree to a leaf is required to insert a new node and still maintain the red black tree properties.

The rules described above are not always applicable and applicability (preconditions of the rules) is determined by the color and structural relationships of other nodes in relation to the current node. For example, for a color flip (rule 1) to be applicable the current node must be black and its two children must be red (For further details, please see [9]). Rule 1 is always applied first (if applicable) and if this results in two red nodes in sequence, either rule 2 or 3 is applied to correct the problem. Similarly after applying rule 4 to insert a new node, rules 2 and 3 are applied to correct any problems. To determine what rules are applicable at a node, we have to find (1) the links of the current node, i.e., identify the nodes that are the children, parent and siblings of the current node, and (2) the color of each of these nodes.

3. TEACHING RED BLACK TREES

Typically, students are first introduced to the properties of red black trees and then taught the algorithm for insertion and later, deletion. A standard way of evaluating whether students have learned the insertion algorithm is for the instructor to provide an input sequence of data and have the student construct the equivalent red black tree while (optionally) showing all the steps taken to arrive at the solution. This evaluation can be performed either electronically (online) or on paper. In our experience, students have generally not performed well on these questions. Red black tree algorithms have always been problematic for students to learn. Past approaches to teaching red black trees (or other balanced trees such as the AVL tree) and the associated algorithms have assumed that the students had difficulty applying these rules and have focused on the mechanics of applying the rules, specifically the single and double rotation rules. These approaches have included:

- textbook exercises to learn how to apply a rule (e.g., [9, 4]),
- visualizations to show how the rules are applied. This approach has been used with many data structures including balanced trees [6, 5], and
- animations to show the steps of the application of the rules [8, 7].

Our experience is that even with the aid of these approaches, students still have difficulty with solving the standard problem, i.e., creating a red black tree from a series of input values. Extensive classroom discussions with students along with analyses of exam questions indicate that the students successfully learn the mechanics of applying the rules (color flip, single and double rotation, simple insertion) but have difficulty in determining when each should be applied. This has led us to identify the following problems:

1. *identification of the current node* when iteratively traversing the tree and applying the rules (Figure 1),
2. *selection of the rule* to be applied at the current node, and
3. *application of the rule* correctly, depicted by the left to right transitions for each of the trees in Figure 1.

Based on this list, we developed a new approach for teaching top-down insertion in red black trees. The earlier approaches focused on the specific mechanics of the color flip, single and double rotation operations. Our approach focuses on (1) the recognition of when each of these rules is valid, i.e., the recognition of the localized context at each step, and (2) the sequence of steps that must be executed to apply one or more rules on the tree.

3.1 A New Approach: The Granularity Approach

In contrast to previous approaches, our approach which we call the *granularity* approach (1) breaks down the algorithm and the accompanying insertion exercises into a combination of smaller exercises so that students need to follow explicit steps to (1) identify the current node, (2) select the rule to be applied, and (3) apply the identified rule at the current node. This explicitly separates out the identification of the current node and selection of the applicable rule from the the application of the rule itself.

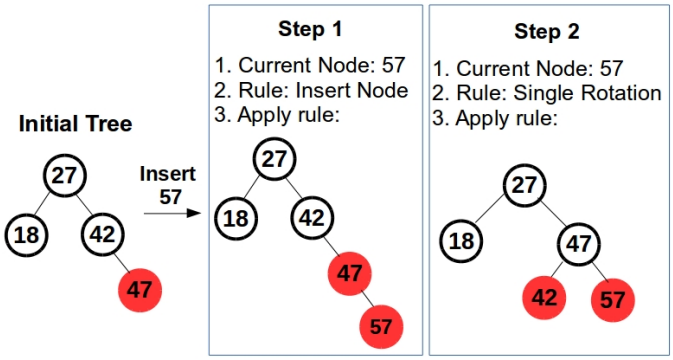

Figure 2: An Exercise Using The Granularity Approach: insertion of 57 into the tree in 2 steps.

3.1.1 *An Example Exercise Using The Granularity Approach*

An exercise using the granularity approach is shown in Figure 2. To simplify the discussion, we assume that the values stored in the nodes of the tree are integers ranging from 1-100. Students are provided with a red black tree and are then asked to apply the top-down insertion algorithm to add a single number (*57*). They are required to show all the steps of their work to reach a solution, with each step representing one single change to the tree from the previous step. In the example in Figure 2, to insert number *57* in the tree, students have to go through two steps to reach the solution. First, they have to apply a *simple insertion* on the *Initial Tree*, that results in the tree in *Step 1*. Secondly, they have to correct the tree from *Step 1* by applying the *single rotation* rule, as shown in *Step 2*, thus arriving at the final correct solution.

We developed granularity based exercises to provide practice for the use of the five rules of top-down insertion, namely color flip, single rotation, double rotation, simple insertion and color the root black. The exercises varied in the context for the selection of each rule. For example, we created exercises where students could apply the color flip rule with the current node being (1) the root of the tree, (2) the left child of the root, and finally (3) the right child of the root. Similarly, we designed exercises for the other rules of top-down insertion.

3.2 Teaching The Granularity Approach With A Tutoring System

We implemented our approach both in the classroom (lecture) and in laboratory exercises. This section describes how the students use the laboratory exercises. The laboratory exercises were provided through an intelligent tutoring system (RedBlackTree Tutor) that we developed using the Cognitive Tutor Authoring Tools (CTAT) [2]. The RedBlackTree Tutor is a web based tutor that students can use to work through the exercises.

Figure 3 shows a sample problem from the RedBlackTree Tutor. In the left part of the figure students are given an initial tree and a single value (a number) to insert. The right part shows an empty tree that allows for many possible transformations (both correct and incorrect). The student now has to provide the answer for the step, and the answer requires that she identify the current node by inserting its

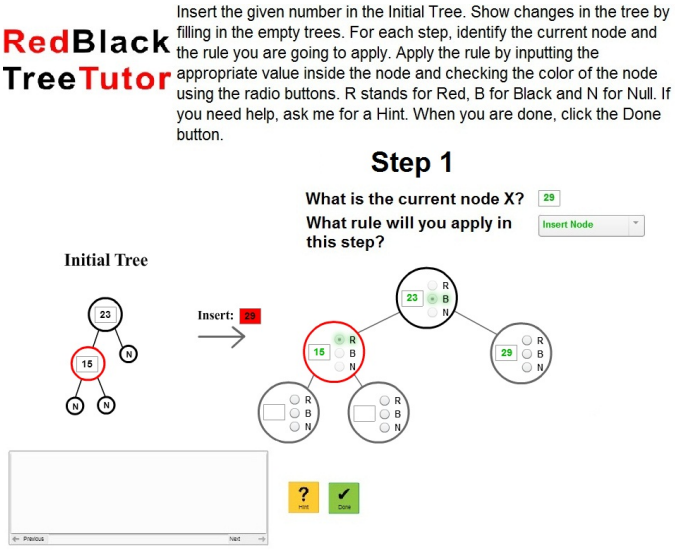

Figure 3: The RedBlackTree Tutor: Left to Right - (1) the problem, (2) solving the problem (simple insertion)

value in the textbox next to "What is the current node X?" question. In addition, the student is asked to select the rule to be applied in that step by choosing one of the five rules from the drop-down menu next to the "What rule will you apply in this step?" question. Lastly, the student is asked to apply the rule by filling in the empty tree. The student can input the appropriate value of the nodes by filling in the textboxes inside each node, and they can specify the color of the node by clicking the 'R', 'B' and 'N' radio buttons which stand respectively for 'Red', 'Black' and 'Null'.

The tutor provides feedback by color highlighting whatever the student enters in a textbox, or selects from a radio button or a drop-down menu. A green color indicates that the student answer is correct, while a purple color signals that the answer is incorrect. In addition, if the student is having difficulties in identifying the current node or rule at a particular step, the student can ask for hints from the tutor.

The RedBlackTree Tutor imposes ordering restrictions on the problem solving path of the student. The first restriction requires the student to provide the correct answer for the current step before moving to the next step. The second restriction is imposed within a particular step, and requires students to answer the two questions at the top, namely identify the current node and select a rule, before going on to applying the rule. The tutor will not allow the student to work on the application of the rule before completing the identification questions correctly. The order restrictions make the students follow the granularity approach, while separating (1) steps from each other, and (2) the identification and selection parts of the problem from the application part.

4. EXPERIMENT DESIGN

We evaluated the granularity approach on the students in our data structures class during the fall semester of 2014. Students are introduced to red black trees during week 8, after they have covered binary trees and binary search trees.

This is the first balanced tree data structure that they will have seen. There were sixteen students in the class, mostly computer science and computer engineering majors and all the students participated in the study. We only analyzed the results for 12 of the students because 4 students had worked and practiced on additional problems on their own in the period between steps 2 and 3 of the evaluation process described below. Their results were discarded from the overall evaluation because we could not disambiguate between the effects of using the RedBlackTree Tutor and the work that they did on their own.

The evaluation process followed the following steps:

1. one week of lecture to cover red black trees,
2. in the following week, a pre-test of 25 mins followed by,
3. a 1 hr session with the RedBlackTree Tutor, and
4. two days later, a 25 min post-test.

4.1 Step 1: Lecture

At the beginning, the class had one week of lecture time (2.5 hours) where red black trees were covered. Students were expected to read the description of the algorithm outside the class time and any misunderstanding or confusion was addressed and discussed in class. During this time the applicability requirements for each rule were described and discussed. In the second half of the week, the students were divided into groups of 3-4 students and were asked to apply the insertion algorithm to a list of 16 values and create the corresponding red black tree showing the tree after every iteration. Each group presented their answers to the class and they compared their solutions. Differences between solutions were discussed and resolved in class.

4.2 Step 2: Pre-Test

In the following week, the students took a 25 min pre-test. Students were asked to show all the steps of their work, and for each step, to identify the current node, select the applicable rule and apply the rule. The pre-test had 4 granularity based exercises that separately tested the students ability to select and apply a double rotation, color flip, single rotation and simple insertion rule respectively. There was a fifth question that tested the students ability to solve problems that required the selection and application of a combination of rules. For this question the students were asked to create a red black tree and sequentially insert the given numbers in the tree.

4.3 Step 3: Working With The RedBlackTree Tutor

We developed 20 granularity based exercises for use with the RedBlackTree Tutor. The exercises were divided into two problem sets that were identical in the rules that were required for each exercise but differed in the numbers used in the initial trees and the numbers to be inserted. In each set there were 3 exercises where students could perform a simple insertion, 3 exercises where they could perform a color flip, 2 for single rotation and 2 for double rotation. In the first problem set, the types of exercises were presented in the order listed above, while in the second problem set, the exercises were mixed so that there was no particular order to their appearance.

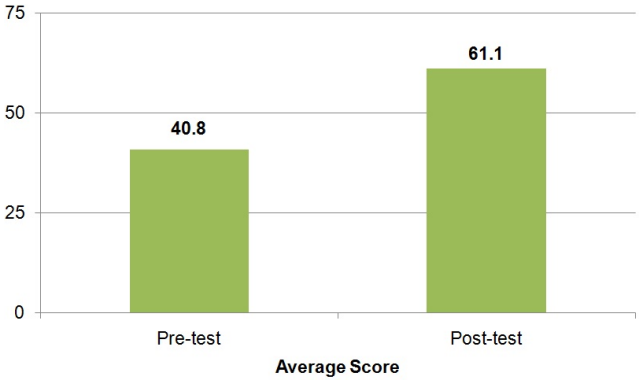

Figure 4: Class performance in pre-test and post-test

4.4 Step 4: Post-Test

Two days later the students had a 25 minute post-test. The exercises for the pre-test and post-test were the same, except that they differed in the numbers used for the initial red black trees and the numbers used for insertion.

5. EXPERIMENTAL RESULTS

The pre-tests and post-tests were graded and scored based on the correctness of each part of each step. We broke down each step in the exercises into 3 parts - identification of the current node, selection of the applicable rule, application of the rule - and graded the answers accordingly. As much as possible, we graded the parts independently. For example if a student started with an incorrect tree from a previous tree but then correctly identified the current node, applicable rule and applied the rule she would be given full credit for all the parts. A part is considered missing when there is no information, e.g., the student does not identify the current node or rule or skips the application of a rule. A missing part, or a missing step can mean (1) the student knows how to fill in the part/step but they did not write it down, or (2) the student does not know how to fill in the part and leaves it blank.

The average score in the pre-test was 40.8 (out of 75) and it improved to 61.1 in the post-test (Figure 4). All 12 students improved their score in the post-test with 8 students having an improvement of 30.0% or more. The maximum percentage increase from the pre-test to the post-test was 169.2% while the standard deviation dropped from 12.2 in the pre-test to 10.2 in the post-test. Overall the average class score increased by 49.6% from the pre-test to the post-test.

The improvement in average scores between the pre-test and post-test is explained by a an overall decrease in the number of incorrect and missing parts, and an increase in the number of correct parts. The number of incorrect parts decreased from 11.67 in the pre-test to 3.92 in the post-test (Figure 5) while the number of missing parts dropped from 22.5 to 10.0 and the number of correct steps increased from 40.83 to 61.08.

We analyzed the scores of the students in (1) identifying the current node at a particular step, (2) selecting the rule, and (3) applying the rule, between the pre-test and post-test. Figure 6 shows that the scores for node identification improved from 10.5 to 18.83 while the scores for rule selec-

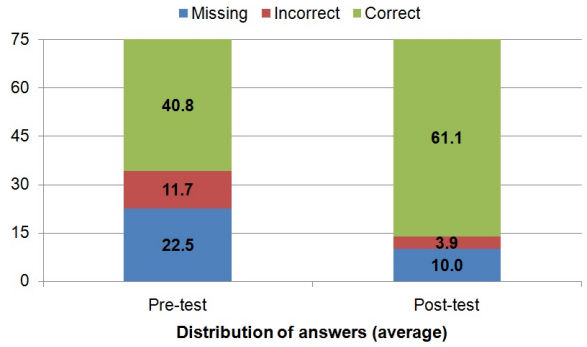

Figure 5: Distribution of the answers for each part in the pre-test and post-test

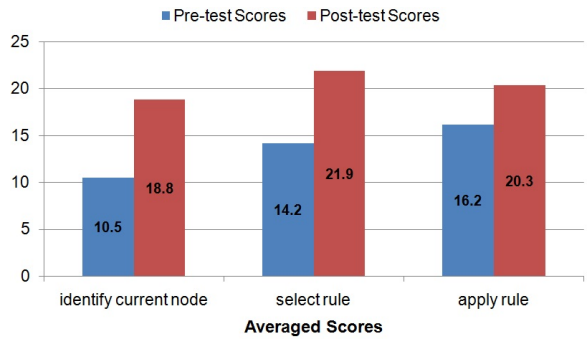

Figure 6: Scores for each part from pre test to post test

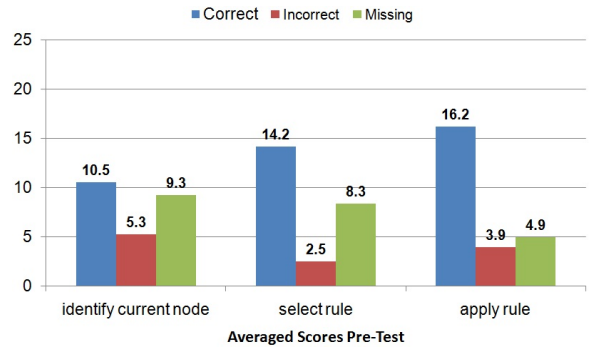

Figure 7: Pre-test scores for each part broken down by correct, incorrect and missing categories

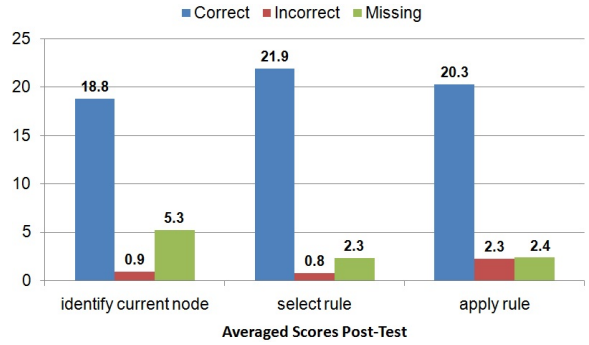

Figure 8: Post-test scores for each part broken down by correct, incorrect and missing categories

tion improved from 14.17 to 21.92 and the scores for rule application improved from 16.2 to 20.3. Thus the scores for node identification increased by 79.33%, rule selection increased by 54.7% and rule application improved by 25.73%.

The data from the pre-test and post-test allowed us to determine the main issues students face when learning the top-down insertion algorithm for red black trees. Figure 7 shows a breakdown of the scores in the pre-test in the areas of identifying the current node, selecting a rule and applying the identified rule correctly. The maximum score for each part is 25. The pre-test shows that following the week of lecture time, the students are best at rule application, followed by rule selection. They are weakest at identifying the current node where the answers have the highest number of missing and incorrect parts. These results support our hypothesis that students have the most difficulty in identifying the current node when learning top-down insertion in red black trees. The data also supports our hypothesis that students have more difficulty in selecting the applicable rule than in applying a selected rule.

Figure 8 shows a similar analysis of the post-test data with 25 being the maximum possible score for each part. The scores in all three parts (node identification, rule selection, rule application) showed substantial improvement from the pre-test so that they show similar values and the answers are mostly correct. The improvement in node identification comes from a drop in the number of incorrect parts and a corresponding increase in correct parts. The data shows that the RedBlackTree tutor has helped students learn how to correctly (1) identify the current node, (2) select the applicable rule and (3) apply the selected rule.

5.1 Discussion

The pre-test data shows that the students had difficulty in applying the top-down insertion algorithm even after a week of lectures that included examples and group work on practice problems. We believe that this is one of those instances where learning the process effectively requires a significant amount of practice. The data shows that after a week of lectures the students had learned the process of applying the rules far more effectively than they learned the process of selecting a rule. The students were able to correctly identify the current node in approximately 40% of the cases while they were able to correctly apply the rule in 65% of the cases.

The data we have collected shows that we may have been incorrectly approaching how we teach red black trees, and perhaps all balanced trees. It will take more experimentation and analysis before we can determine the cause of the problems, but in the meantime the granularity approach has been shown to effective at solving the problems.

6. CONCLUSION AND FUTURE WORK

This paper has described a new approach for teaching students about insertion in red black trees. We have described its implementation in an intelligent tutoring system, RedBlackTree Tutor and evaluation in a data structures course in the fall semester of 2014. The initial analysis of the data

collected in the pre-tests and post-tests of this first iteration showed that students had significant learning gains of 49.6% in performance. The data showed an overall performance improvement, individually and in the class overall. We also noticed a significant decrease in the number of incorrect and missing parts from the pre-test to the post-test and a corresponding increase in the number of correct parts.

The data we collected provides insights as to the source of the difficulties for students learning insertion in red black trees. The data indicated that the main problems students face are in correctly identifying the current node and selecting the applicable rule, rather than in applying a selected rule. We intend to extend our approach to include deletion in red black trees and also to carry out a similar evaluation in the spring semester of 2015.

7. REFERENCES

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. ACM/IEEE Computing Curricula 2001 Final Report. `http://www.acm.org/sigcse/cc2001`, 2001.

[2] V. Aleven, B. M. Mclaren, J. Sewall, and K. R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2):105–154, 2009.

[3] D. Chinn, P. Prins, and J. Tenenberg. The role of the data structures course in the computing curriculum. In *Proceedings in The Journal of Computing Sciences in Colleges*, volume v19 #2, 2003.

[4] W. J. Collins. *Data Structures and the Java Collections Framework*. McGraw-Hill, 3rd edition, 2011.

[5] D. Galles. Data structure visualizations. `http://www.cs.usfca.edu/~galles/visualization`.

[6] S. Ha. VisuAlgo. `http://www.comp.nus.edu.sg/~stevenha/visualization`.

[7] J. Kloss. Animated data structures. `http://www.cs.jhu.edu/~goodrich/dsa/trees`.

[8] W. C. Pierson and S. H. Rodger. Web-based animation of data structures using jawaa. *ACM SIGCSE Bulletin*, 1998.

[9] M. A. Weiss. *Data Structures & Problem Solving Using Java*. Pearson Education Inc., 3rd edition, 2011.