An Integrated and “Engaging” Package for Tree Animations

Guido Rößling, Silke Schneider
Technische Universität Darmstadt
Darmstadt, Germany

roessling@acm.org

Abstract

This paper describes a prototypical system that combines several aspects of engagement as defined in (Naps et al. 2003) for the topic of tree and tree algorithm animations.

1 Introduction

The ITiCSE 2002 Working Group on *Improving the Educational Impact of Algorithm Visualization* proposed the following engagement taxonomy for active AV use (Naps et al. 2003):

1: No viewing – AV content is not used at all;

2: Viewing – AV is used, but the user’s involvement is centered on consuming the content or controlling the progress (forward / pause / faster / ...);

3: Responding – the user is asked certain questions while the content is presented, usually asking him or her to predict the next state of the animation;

4: Changing – the user can modify the visualization, usually by specifying the input data set or selecting actions that cause an update of the visualization;

5: Constructing – the user generates a new animation of a given algorithm;

6: Presenting – the user presents a visualization to an audience for feedback and discussion. The content of the visualization does not have to be created by the user.

The Working Group also stated two hypotheses: first, that there is no significant difference regarding learning outcomes between the two lower levels; and second, that there will be a significant difference in learning outcomes when comparing any other “higher” level with a “lower” level. Thus, each step in the hierarchy (apart from the first two) is supposed to increase the possible improvement in learning outcomes.

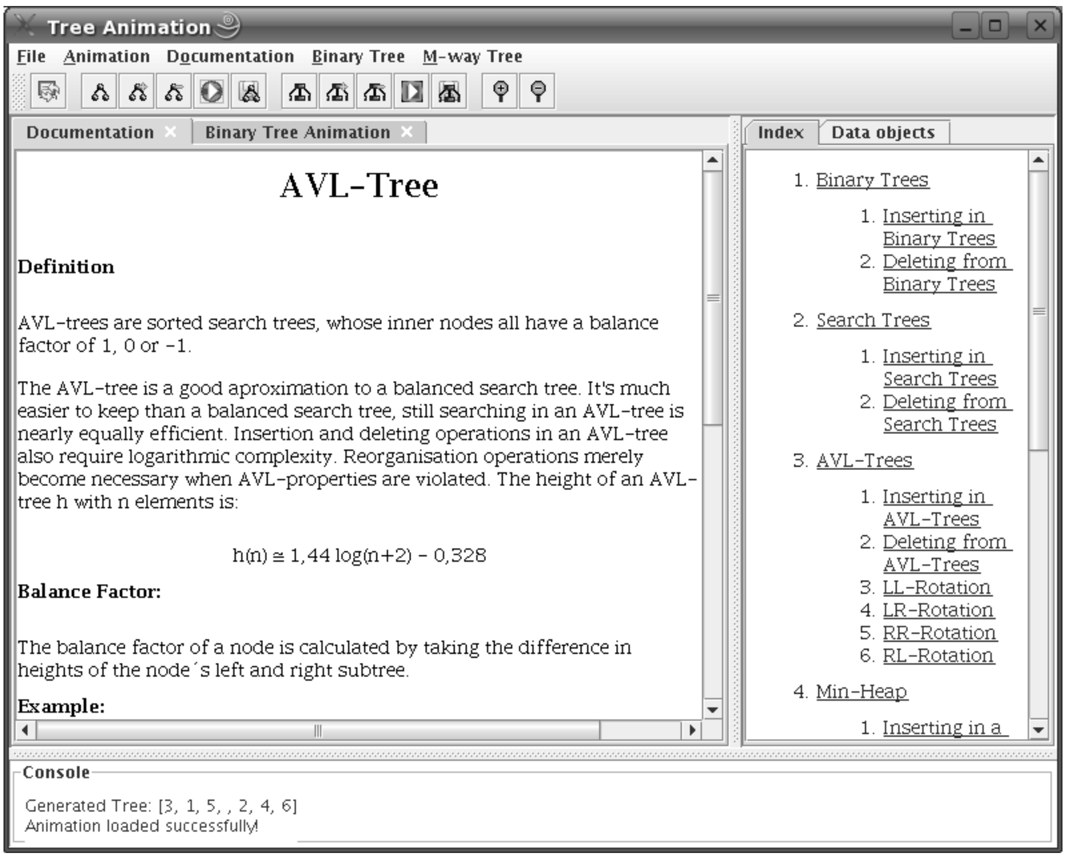
In this paper, we present a new Java application for animating trees and tree algorithms. Apart from the engagement levels 1 and 2 supported by all AV tools, the application also supports levels 3 (responding) and 4 (changing), and can be used for level 6 (presenting).

2 Visualizing and Creating Tree Animations

Figure 1 shows the welcome screen for the application. The application consists of four areas. At the top is the control area, containing the application’s menu bar and (draggable) tool bar for creating or modifying the current trees. The main part of the window is taken up by the *documentation* and *animation area*, which shows the current content.

The right side contains an index of the web pages as a reference. This page is essentially identical to the welcome page initially shown in the documentation tab of Figure 1. The *Data objects* view provides access to the current instances of the two supported tree types, binary and m-way trees. Finally, a console at the bottom of the window displays the output of the operations.

The buttons in the toolbar can be used for performing tree operations. The first and the last two buttons shown in Figure 1 can be used to load a new animation, and zoom in and out of the animation, respectively. The two groups of five buttons each are used for creating a new tree, inserting a new key, removing a key, showing the generated animation, and saving the animation to disk. The first set of buttons is for binary trees, while the second group is for m-way trees, as indicated by the node form used for the icons.



AVL-Tree

Definition

AVL-trees are sorted search trees, whose inner nodes all have a balance factor of 1, 0 or -1.

The AVL-tree is a good aproximation to a balanced search tree. It's much easier to keep than a balanced search tree, still searching in an AVL-tree is nearly equally efficient. Insertion and deleting operations in an AVL-tree also require logarithmic complexity. Reorganisation operations merely become necessary when AVL-properties are violated. The height of an AVL-tree h with n elements is:

$$h(n) \cong 1{,}44 \log(n+2) - 0{,}328$$

Balance Factor:

The balance factor of a node is calculated by taking the difference in heights of the node´s left and right subtree.

Example:

1. Binary Trees
   1. Inserting in Binary Trees
   2. Deleting from Binary Trees
2. Search Trees
   1. Inserting in Search Trees
   2. Deleting from Search Trees
3. AVL-Trees
   1. Inserting in AVL-Trees
   2. Deleting from AVL-Trees
   3. LL-Rotation
   4. LR-Rotation
   5. RR-Rotation
   6. RL-Rotation
4. Min-Heap
   1. Inserting in a

```
Generated Tree: [3, 1, 5, , 2, 4, 6]
Animation loaded successfully!
```

Figure 1: The tree visualization application with index and AVL tree documentation

The application currently supports the following types of trees:

- binary trees,
- binary search trees,
- AVL trees,
- Heaps, implemented as *min* or *max heaps*,
- and m-way search trees with the implementation of a B tree.

The trees were modeled as described in the literature, e.g. by Goodrich and Tamassia (2005). It is easy to add new implementations, for example for red-black trees. To do so, the implementation has to inherit from the appropriate class - for red-black trees, this is the class *tree.SearchTree*. Additionally, some of the methods implemented by the super class may have to be overridden. This usually concerns the methods for inserting or deleting keys, reordering the tree after operations, and in some cases, for adapting how the nodes are painted.

The user is free to create a new tree type by selecting either binary or m-way trees from the menu. He can then decide whether documentation shall be included in the display, and whether interactive questions should be asked during execution, as described by the 2002 ITiCSE Working Group (Naps et al. 2003). The user then works with the tree using the buttons shown in the toolbar of Figure 1 to insert or remove keys.

The “play” button displays the current state of the tree as an animation. The animation is generated in AnimalScript and displayed by a customized front-end of Animal. Figure 2 shows an excerpt of an AVL tree animation. In the Figure, the insertion of the key *19* will cause a *RL(i) rotation* to rebalance the AVL tree. This double rotation is known to be often poorly understood by students. Therefore, the animation incorporates a schematic display of the state of the (sub-)tree before and after the operation, shown on the left side of Figure 2.

Even after the animation is started, the user can continue modifying the tree using the buttons described above. The application creates add-on visualization code for the current animation to reflect the results of the actions taken by the user. This can then be loaded in using the “incremental loading” button shown at the bottom right of Figure 2. Instead of parsing the complete animation, only the added code will be parsed and added to the end of the current visualization, boosting the speed of the display.

The application described so far thus combines the generation of a tree according to the user’s operations and the stepwise animation of the output. Therefore, this places the application on the levels 2 (viewing) and 4 (changing) of the Working Group engagement taxonomy (Naps et al. 2003).

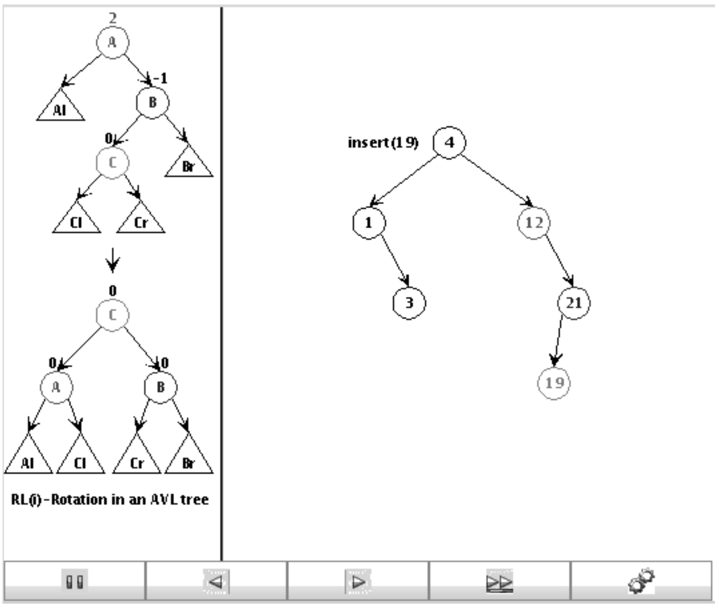


Figure 2: Excerpt of the AVL animation illustrating a *RL(i) double rotation*

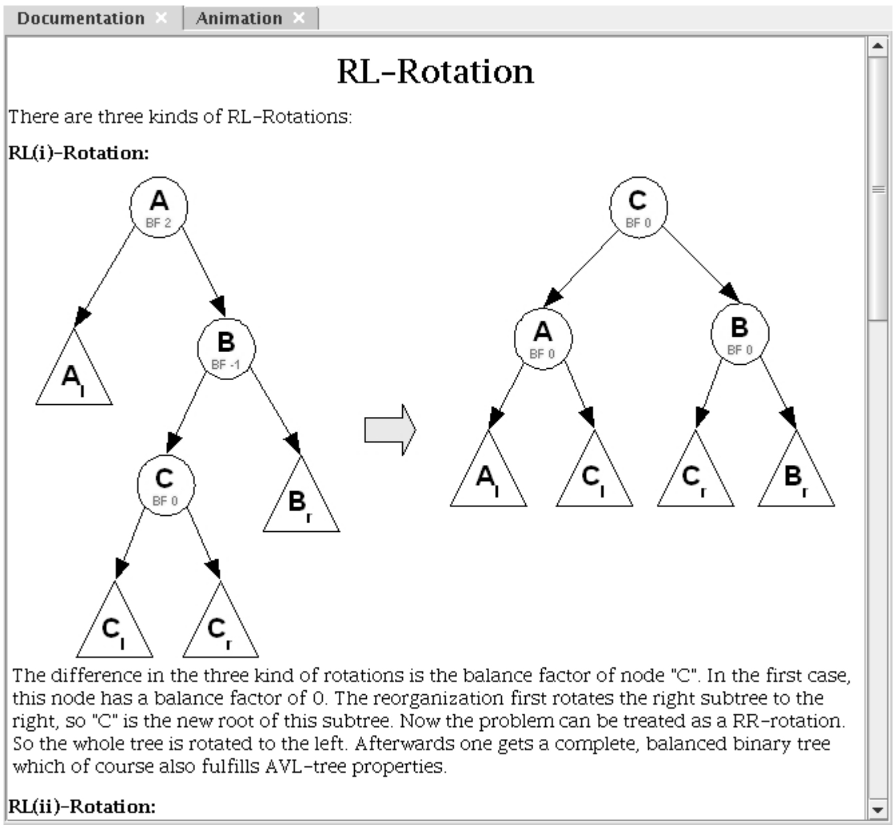
3 Embedded Documentation

Rößling and Naps (2002) stated the importance of incorporating hypertext explanations of the visual display. This documentation is meant to describe the operation of the underlying visualization engine, the mapping of the algorithm to the visualization, and should be “ideally adaptive to the current state of the algorithm”.

The approach up to this stage is similar to some existing AV systems, especially *Matrix-Pro* (Karavirta et al. 2004). However, some of the planned features, especially considering embedded documentation as shown in Figure 2, are not easily implemented in these systems. Basing the work on AnimalScript makes generating content like this relatively easy.

Users of the application can use the built-in documentation provided in HTML format. Figure 3 shows part of the HTML page explaining the *RL(i)* rotation animated in Figure 2. This HTML documentation does not refer to actual values, but provides one page for each possible operation (insertion and removal of tree nodes and all balancing or reordering operations).

Additionally, the user can decide to turn on additional documentation to explain the individual operations. The documentation is adaptive to the current state of the algorithm and also mentions the concrete affected elements for any operation.



RL-Rotation

There are three kinds of RL-Rotations:

RL(i)-Rotation:

The difference in the three kind of rotations is the balance factor of node "C". In the first case, this node has a balance factor of 0. The reorganization first rotates the right subtree to the right, so "C" is the new root of this subtree. Now the problem can be treated as a RR-rotation. So the whole tree is rotated to the left. Afterwards one gets a complete, balanced binary tree which of course also fulfills AVL-tree properties.

RL(ii)-Rotation:

Figure 3: Example documentation: performing a *RL(i)* rotation on an AVL tree

This documentation is included in the animation, to prevent the user from having to switch views. When it appears, it is placed at the top of the animation window. The documentation is currently available for all supported tree types except for B-Trees. It will document both insertion and removal operations in one of four levels of detail:

NO_DOCUMENTATION turns off all documentation features within the animation;

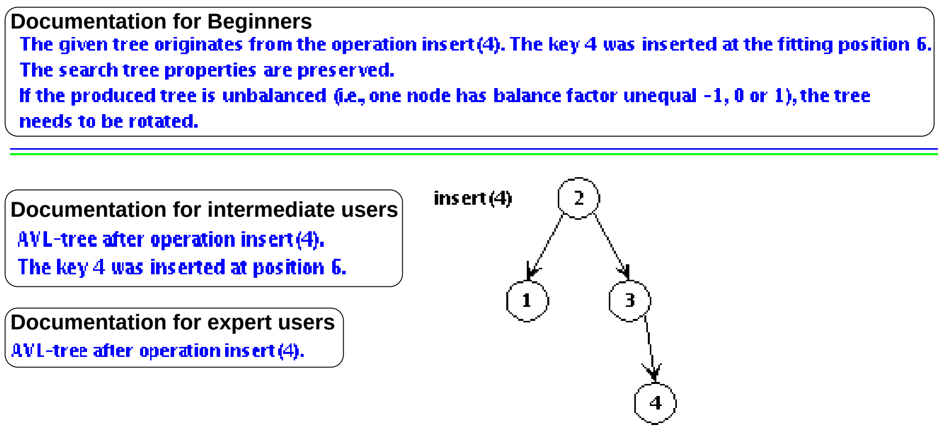
EXPERT offers a single line of rather terse documentation;

INTERMEDIATE offers two lines of documentation for each operation;

BEGINNER offers up to four lines of documentation for each insertion or removal.

Figure 4 shows the three documentation levels (*NO_DOCUMENTATION* is not shown, as it is by definition empty). The basic animation content is the documentation for beginners, with the tree to the right. The documentation for intermediates and experts was grafted to this Figure to allow for easy comparison. The documentation, and indeed the whole application, is prepared for internationalization, and already addresses English and German. Adding more languages is fairly straightforward, as it mainly requires translating the resource text files.

The combination of live tree exploration with embedded *and* external documentation brings the application close to the realm of hypertextbooks described by Ross and Grinder (2002), also addressed by an ITiCSE 2006 Working Group. It also makes it easy to use the application to present tree algorithms to an audience, making the system applicable to level 6 (presenting) of the engagement taxonomy for its (admittedly) narrow focus.



Documentation for Beginners

The given tree originates from the operation insert(4). The key 4 was inserted at the fitting position 6. The search tree properties are preserved. If the produced tree is unbalanced (i.e., one node has balance factor unequal -1, 0 or 1), the tree needs to be rotated.

Documentation for intermediate users

AVL-tree after operation insert(4). The key 4 was inserted at position 6.

Documentation for expert users

AVL-tree after operation insert(4).

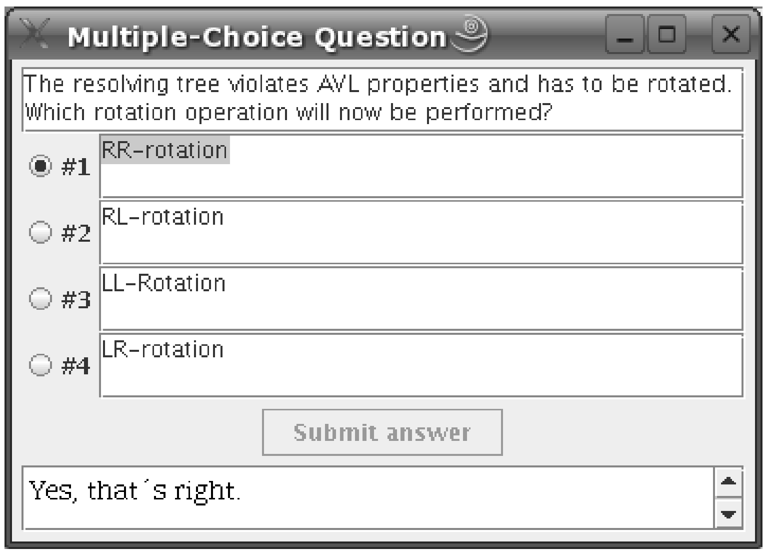
Figure 4: Documentation levels in the animation: beginner, intermediate, and expert

4 Interactive Prediction Support

The *responding* category in the engagement taxonomy expects the animation to be interrupted by questions. These questions will typically prompt the user to predict the next step performed by the algorithm, or ask him or her to describe the current visualization state.

For this end, we have added the *avInteraction* package to the application, as described by Rößling and Häussge (2004). The functionality of this package is similar to the "interactive prediction" supported by *JHAVÉ* (Rößling and Naps 2002). The avInteraction package can easily be extended or adapted to other systems. Additionally, it offers other interesting features, such as skipping questions once a specified number of "related" questions were answered correctly.

Figure 5 shows an example pop-up window that prompts the user to predict which rotation will occur in the next step of the AVL tree reorganization. Currently, only AVL trees and B-trees incorporate interactive prediction, although this can be changed easily.



The resolving tree violates AVL properties and has to be rotated. Which rotation operation will now be performed?

[✓ #1] RR-rotation

#2 RL-rotation

#3 LL-Rotation

#4 LR-rotation

Yes, that´s right.

Figure 5: Interactive Prediction: determining the correct type of rotation

5 Summary and Further Work

In this paper, we have presented an extensive application for creating tree visualizations. Based on the underlying Animal system, generating the animation code is fairly simple. The application merges several requirements from past Working Group reports and research papers: both "static" and "live" documentation are included, and interactive questions can be activated to make the learner's session more engaging. Finally, the extensive collection of HTML pages describing the underlying data structures and algorithms makes this a promising learning tool for trees.

We have incorporated some new ideas into this prototype. First, this is one of the first systems that we are aware of that uses the actual values in its built-in documentation, and supports different levels of documentation detail. The ability to add code to the visualization without having to re-parse from the beginning is also a new features, and definitely a premiere for the Animal system.

In the future, we want to further explore the connection between the ideas used in this system and the hypertextbook concept (Ross and Grinder 2002). We also plan to add some of the tried and tested features, especially the "incremental loading", into the Animal AV system.

We are also looking for educators interested in trying out the application. We would especially appreciate colleagues who are willing to take part in an evaluation of the system regarding learning outcomes, as our teaching obligations currently do not include access to the data structures course, and therefore keep us from the key clientele.

References

1 Michael Thomas Goodrich and Roberto Tamassia. *Data Structures & Algorithms in Java.* Wiley & Sons, 2005.

2 Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. MatrixPro - A Tool for Ex Tempore Demonstration of Data Structures and Algorithms. In *Proceedings of the Third Program Visualization Workshop, University of Warwick, UK*, pages 27–33, July 2004.

3 Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin*, 35(2):131–152, 2003.

4 Rockford J. Ross and Michael T. Grinder. Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web. In Stephan Diehl, editor, *Software Visualization*, number 2269 in Lecture Notes in Computer Science, pages 269–284. Springer, 2002.

5 Guido Rößling and Gina Häussge. Towards Tool-Independent Interaction Support. In *Proceedings of the Third Program Visualization Workshop, University of Warwick, UK*, pages 99–103, July 2004.

6 Guido Rößling and Thomas L. Naps. A Testbed for Pedagogical Requirements in Algorithm Visualizations. *Proceedings of the 7th Annual ACM SIGCSE / SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002), Århus, Denmark*, pages 96–100, June 2002.