Comic Strips for Algorithm Visualization*

Henning Biermann† Richard Cole‡

February 8, 1999

Abstract

This paper presents visualizations of binary search trees and splay trees. The visualizations comprise sequences of figures or frames, called *comic strips*. Consecutive frames are viewed two at a time to facilitate user (viewer) understanding of the algorithm steps. The visualizations are implemented in Java to facilitate their wide use. This paper explores several other considerations in the design of instructional visualizations.

1 Introduction

The potential of dynamic visual presentations of combinatorial algorithms has been manifest for at least two decades. One early and impressive demonstration was the sorting movie of Backer and Sherman [BS82]. A subsequent landmark, which delineated the design of animation systems, was the Zeus system [Bro91] of Brown (also see his thesis [Bro83, Bro87]). Over the years, it has inspired a still growing number of animation systems (Tango [Sta90], XTango, Polka, Leonardo [CDF+97]). The advent of Java and the World Wide Web has caused a further burgeoning of interest in the topic, including new developments of some of the above systems and other new systems, e.g. JAWAA [PR98], Jeliot [HPS+97], JSamba (see http:// www.cc.gatech.edu/gvu/softviz/algoanim/jsamba/), JCAT (see http://www.research.digital.com/SRC/ JCAT/). At this point there are also many online animations of specific algorithms. References to many educational animations can be found at http://www.cs.hope.edu/∼alganim/ccaa/ and at the home page for Goodrich and Tamassia's algorithms textbook: http://www.cgc.cs.jhu.edu/∼goodrich/dsa/. Visualizations of binary search trees and splay trees can be found at http://langevin.usc.edu/BST/ (due to Ierardi) and of splay trees alone at http:// gs213.sp.cs.cmu.edu/cgi-bin/splay (due to Sleator).

Nonetheless, the reach and impact of this work has been limited until recently. In part, until the advent of Java, there was significant overhead in installing and using algorithm animation systems. Further, most work focussed on systems for creating animations rather

*This work was supported in part by NSF grant CCR-9503309 and CCR-9800085.

†Courant Institute, NYU, biermann@cs.nyu.edu.

‡Courant Institute, NYU, cole@cs.nyu.edu.

than the animations themselves. In this paper, we focus instead on what features are key to effective educational illustrations of algorithms.

We are interested in combinatorial algorithms, namely the staple of “Algorithms” and “Data Structures” classes. Our approach is to visualize such algorithms as a sequence of views or *frames*. The viewer sees the frames through a window which shows two consecutive frames (which inspired the name “comic strip” for the sequence of frames). Our perspective is that a combinatorial algorithm, running on a given input, performs a sequence of steps or actions, and that to understand the algorithm one needs to understand the individual actions (at whatever level of granularity is appropriate for the algorithm and the student). The natural way to understand an action is to see the state of the system (a data structure for the algorithms we have in mind) both immediately before and after the action. Thus it is natural to view consecutive frames simultaneously. The viewer is able to advance and wind back the window across the frame sequence one frame at a time. In this setting we consider it unnatural to have warping or animation from one frame to the next, and thus each frame in our visualizations is a separate unchanging entity. Another consideration is that this approach emphasizes the discrete nature of the algorithms. This also leads us to prefer the name *Algorithm Visualization* over the more common *Algorithm Animation*.

To make our discussion concrete, in Section 2 we describe our visualizations of Binary Search Trees and Splay Trees. In Section 3, we discuss the choices we made and the rationale for our approach, and in particular we explore the considerations underlying the “Comic Strip” approach to visualization. In Section 4, we give an overview of the algorithm description of which the visualization is just a part.

2 Binary Search and Splay Tree Visualizations

The Binary Search Tree and the Splay Tree are presented as one visualization. At any stage, there is a current binary search tree, to which the viewer can apply a choice of standard operations:

- Binary Search Tree Operations
  search(x), insert(x), delete(x)

- Splay Operations
  splay(x), splay-insert(x), splay-delete(x)

Each operation is demonstrated by means of a step by step visualization.

We provide a choice of initial trees, including the empty tree. We also have operations insert-fast(x) and delete-fast(x) which show fewer steps of the visualizations (for insert(x) and delete(x) resp.), both to assist with creating trees and for the viewer who needs less detail.

The sequence of frames for search(x) is illustrated by the example in figure 1.

At each step, the viewer sees a window containing two successive frames. In effect, the visualization moves this window across the sequence of frames. The viewer controls the

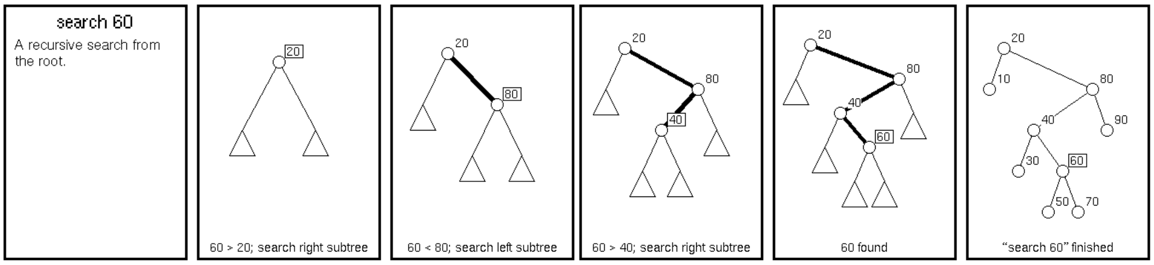

Figure 1: Example of Binary Search

sequencing of frames with NEXT and PREVIOUS buttons that move the window one frame to the right and left, respectively.

The reason for showing two frames at a time is to allow the viewer to observe how the data structure at the current step, shown in new frame, is obtained from the data structure at the previous step, shown in the preceding frame. The alternate approach of showing the frames one at a time forces the viewer to remember the exact form of the previous configuration, which seems an unnecessary burden on the viewer. The utility of showing more than one frame at a time was already observed by Naps and Bressler [NB98]; they provide an environment for showing multiple frames, which could be successive views of a data structure, or might be alternate views of the same algorithm configuration. Our perspective, which we discuss further in Section 3, is that it is essential to present successive frames within one window.

As can be seen, our visualization shows just the portion of the access path that has been traversed. The remainder of the tree is shown as a set of subtrees, each represented by a triangle. The purpose is to keep the viewer's attention on the portion of the data structure that is influencing the operation (namely, the item at the end of the traversed portion of the access path). It also shows what information is known to the operation at hand. In addition, this visualization seeks to emphasize the recursive nature of the search, by showing the zone in which the search is to continue as a single subtree.

The actual visualization is in color and in consequence the key features are more clearly distinguished than in the above figure.

The visualization of the insert operation shows fewer steps, as it assumes the search operation is understood. This is illustrated by the example in figure 2.

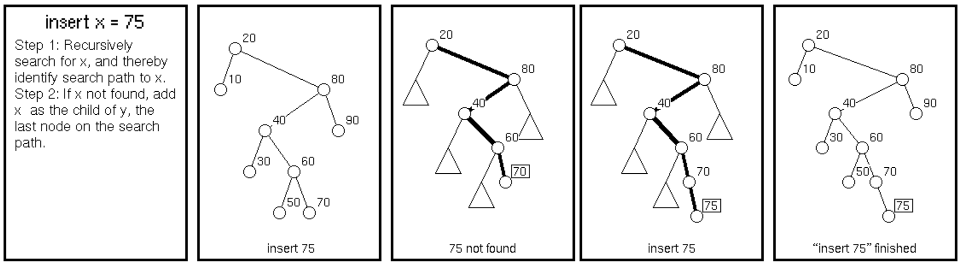

Figure 2: Example of Binary Search Tree Insertion

The visualization of the delete operation makes the same assumption, but needs to show slightly more intermediate steps. One case is illustrated in the example in figure 3.

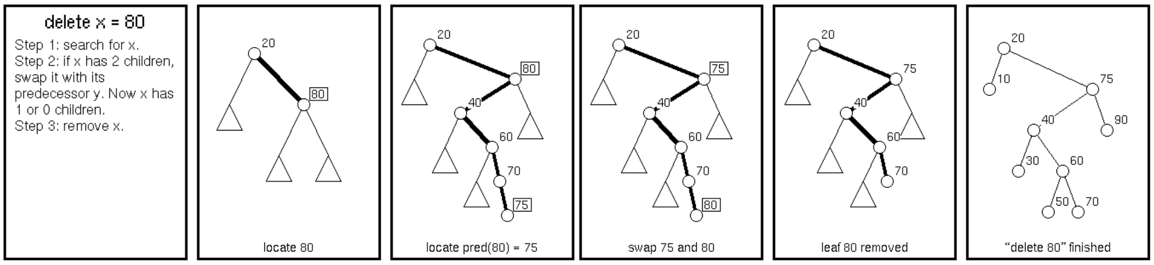

Figure 3: Example of Binary Search Tree Deletion

In order to build or change a tree quickly we allow the insert-fast operation to take more than one argument, namely insert-fast($x_1, x_2, \cdots, x_k$). This is visualized by inserting the items in the argument list one after another in successive frames, one frame per insertion. No doubt it would be convenient to have a one-step visualization of the full sequence of insertions, but we have yet to implement this.

We turn now to the visualization of splay operations. Recall that a splay tree is just a binary search tree. In the operation splay(x), the item x is first located (by the search(x) operation) and then it is moved to the root by a sequence of double rotations and possibly one single rotation. The visualization focuses on the nodes on the access path, showing the remainder of the tree as a collection of subtrees, each represented by a triangle as before. In the visualization, the first step identifies the nodes on the access path; it then shows in turn each rotation that occurs as the accessed item is moved to the root. This is illustrated in the example in figure 4.

Again, the actual visualization is in color. In particular, for each rotation operation, the three nodes on the access path being rotated are all colored green; each of the four subtrees hanging from these three nodes is also colored a distinct color (some of these subtrees may comprise individual nodes from the portion of the access path already traversed, plus appropriate subtrees). This coloring allows the viewer to readily check that the appropriate rotation has indeed been performed (in our experience, in examining figures of whole trees, it is hard to superimpose the structure of the subtrees for the rotation at hand on the flat representation of the tree, whereas the coloring causes this structure to stand out). While the black and white patterns in our illustration provide the same visual cues, they are not quite as immediate.

One might expect to have only every third frame of those we show present in a visualization. But our perspective is that to show a step, one needs to show the data structure configuration before and after each operation. Thus for each rotation there are two frames showing the tree immediately before and after the rotation, with common structures colored consistently. We chose to introduce a third frame between each pair of before/after frames to highlight the switch of focus from one group of three nodes on the access path involved in a rotation to the next group; this third frame also provides a helpful punctuation.

As a further aid to the viewer, for each rotation, we provide an optional pop-up window which shows the rule being applied in this rotation (zig-zig, zig-zag, etc.). One such rule is shown in figure 5.

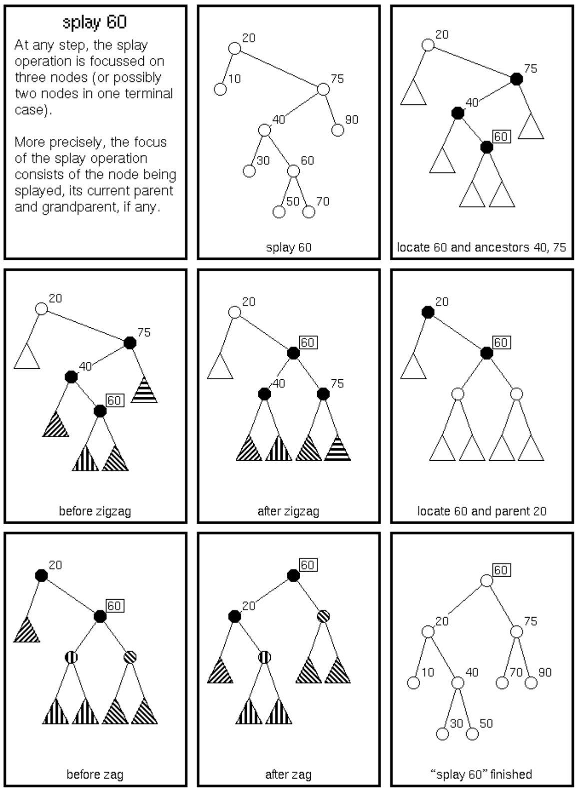

Figure 4: Example of Splay Operation

Again, the splay-insert and splay-delete operations show a shorter sequence of frames as they assume the basic splay operation is understood. An example is illustrated in figure 6.

These visualizations can be found at

http://www.cs.nyu.edu/algvis/

The visualizations are complemented by illustrated textual explanations of the data structures and the algorithms, together with a small set of exercises for the reader.

3 The Comic Strip Approach

The core of visualization design is the decision regarding which actions to illustrate, i.e. partitioning the operation into a sequence of smaller changes. The designer then needs to devise illustrations for the individual actions. The design of these illustrations has been based on the following guidelines:

1. Focus on the current action.

   Hide unnecessary information; specifically, hide data that is not part of the current action or emphasize data that is involved in the current action.

2. Avoid forcing the viewer to remember frames no longer on display.

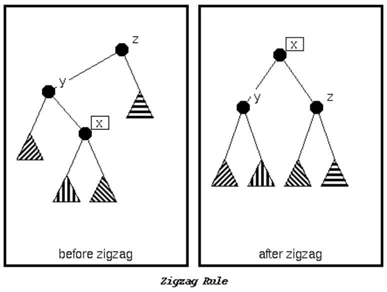

Figure 5: Pop-up Window for Zigzag rule

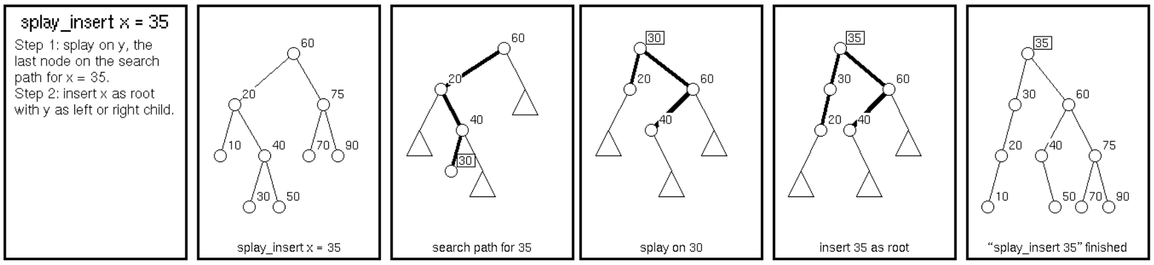

Figure 6: Example of Splay-Insert Operation

In particular, show both a "before" and an "after" view for each action that is being visualized.

3. Provide full viewer control of frame changes.
   (a) Each change of frames occurs only when the viewer requests it (e.g. with a NEXT button).
   (b) Make the sequence of actions reversible, certainly a frame at at a time, and possibly at larger granularities (e.g. with a PREVIOUS button).
   (c) Enable viewing of the action at various granularities.
4. Provide built-in examples (created by the visualization designer).
5. Enable user created examples.

Having chosen the granularity of the visualization (i.e. what constitutes a single action), each pair of frames on display needs to explain the action they illustrate. It is our belief that this is done most effectively by minimizing the information on display that is not pertinent to this action.

Our second guideline can pull in the opposite direction. It is helpful to the viewer, when seeking to understand the currently displayed frame(s), if s/he does not have to remember the configuration of a frame that is no longer on display. This has led us to display both a "before" and an "after" view for each action that is being visualized. But this goal may also entail additional information being present; for example, in the splay tree visualization, this goal leads to showing the full access path throughout the access, even though only 3 nodes participate in any given rotation (the full access path is shown because at the end of the splay operation it is natural to display the reorganized tree, and the portion of the tree that has changed is exactly the arrangement of the nodes on the access path).

We seek to reconcile these guidelines by the use of other visual cues. Continuing with the splay tree example, we have argued that the full access path should be shown in all frames, but there are only 3 nodes that matter in any given rotation. A natural compromise is to highlight the nodes that matter, which we do by coloring them with a distinct, bright color.

The choice of "Before" and "After" views is more involved than might appear at first sight. To facilitate our discussion, we introduce a little notation. Let S_i be the ith step (or action) being visualized. Let VB_i denote the view immediately before S_i and VA_i denote the view immediately after S_i. The most appealing situation is to have views VA_i and VB_{i+1} be identical; this is the case, for example, in the visualization of the Binary Search Tree search. However, in the visualization of the splay operation, VA_i and VB_{i+1} differ, for the highlighted nodes are distinct. One approach would be to visualize successive steps in successive views. Thus one would have frames VB_i and VA_i present in one view, and VB_{i+1} and VA_{i+1} present in the next view. One difficulty is that one would like to make it easy for the viewer to verify the transition from VA_i to VB_{i+1}. One option would be to have more than two frames present in a single view, but this does not conform with the goal of having the viewer focus on one action at a time. Another option is to have an intermediate view comprising the frames VA_i and VB_{i+1}. This is the approach we favor. In the splay tree example though, we prefered to introduce another intermediate frame, which serves to better separate the rotation actions.

Our decision to provide viewer control over frame changes instead of selecting a scrolling rate is based on our belief that different viewers will assimilate explanations at different speeds and not necessarily at a uniform rate. Clearly, too, a viewer may wish to reexamine a previously seen frame; hence the PREVIOUS button. In our visualizations the control over granularity is limited to a choice between an operation and its Fast variant (for Insert and Delete on the binary search tree).

4 Algorithm Description

The aim of the on-line material is to provide a self-contained algorithm description. To this end, we have written standard textual algorithm explanations, augmented with examples presented using our binary search tree visualizer. We have also written a small collection of exercises, some of which are most readily solved using the binary search tree visualizer. In addition, the viewer can use the binary search tree visualizer to examine the actions of the algorithm on their own examples.

References

[BN96] M. H. Brown and M. A. Najork. Collaborative active textbooks: A web-based algorithm animation system for an electronic classroom. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 266–275, 1996.

[Bro83] Marc H. Brown. *Algorithm Animation*. PhD thesis, Brown University, Providence, RI, 1983.

[Bro87] M. H. Brown. *Algorithm Animation*. ACM Distinguished Dissertation. MIT Press, Cambridge, MA, 1987.

[Bro91] M. H. Brown. Zeus: A system for algorithm animation and multi-view editing. In *IEEE Workshop on Visual Languages*, pages 4–9, 1991.

[BS82] Ronald M. Backer and David Sherman. Sorting out sorting, 16mm color sound film, 30 minutes, 1981. Shown at ACM SIGGRAPH '81 in Dallas, TX and excerpted in ACM SIGGRAPH Video Review # 7, 1982.

[CDF+97] P. Crescenzi, C. Demetrescu, I. Finocchi, and R. Petresci. LEONARDO: a software visualization system. In *Workshop on Algorithm Engineering*, 1997. See also http://www.dsi.unive.it/∼wae97/proceedings/.

[HPS+97] J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta, and P. Vanninen. Animation of user algorithms on the web. In *IEEE Symposium on Visual Languages*, pages 360–367, 1997. See also http://www.cs.Helsinki.FI/research/aaps/Jeliot/.

[NB98] Thomas L. Naps and Eric Bressler. A multi-windowed environment for simultaneous visualization of related algorithms on the www. In *SIGSCE Technical Symposium on Computer Science Education*, pages 277–281, 1998.

[PR98] Willard C. Pierson and Susan H. Rodger. Web-based animation of data structures using jawaa. In *Twenty Ninth SIGSCE Technical Symposium on Computer Science Education*, pages 267–271, 1998. See also http://www.cs.duke.edu/∼wcp/JAWAA.html.

[Sta90] John T. Stasko. Tango: A framework and system for algorithm animation. *Computer*, 23(9):27–39, September 1990.