A Tool for Sketching and Manipulating Binary Heaps

Brad Vander Zanden and Michael Beeler
Computer Science Department
University of Tennessee
Knoxville, TN 37996
bvz@cs.utk.edu, beeler@cs.utk.edu

ABSTRACT

Data structures are one of the most fundamental topics taught in computer science courses. Instructors often use diagrams to illustrate the operations that are performed on data structures but there is a lack of computerized teaching tools with domain-specific knowledge that could assist an instructor in sketching and manipulating these data structures. This paper introduces a prototype tool that allows users to quickly sketch and manipulate binary heaps. The tool allows optional captions to be associated with the manipulations of the heaps so that students can use this tool for self-directed study. Eventually we hope to extend this tool to work with many different types of data structures, including arrays, lists, trees, and graphs.

Categories and Subject Descriptors

E.1 [**Data Structures**]: *trees*; K.3.2 [**Computers and Education**]: Computer and Information Sciences Education

General Terms

algorithms

Keywords

binary heaps, animation, instruction

1. INTRODUCTION

Instructors in Computer Science often present new data structures to students using a combination of pseudo-code and drawings on a whiteboard. As instructors explain the algorithms associated with the data structure, they will rapidly change the drawings to represent changes within the structure. A student attempting to follow the instructor's lecture and take quality notes on the topic may quickly find they are doing neither.

Our research group is designing a data structure sketching tool that aims to change this dynamic. An instructor using this tool can rapidly sketch a data structure, type in any needed values, and use pre-supplied animations to provide a step-by-step demonstration of how the structure is changed as various algorithms are performed on it. Instructors can save their examples and students can later review them or build their own examples and test various algorithms on the newly sketched structures. Captions are provided with each animation step so that the student can understand what is happening, even without having the instructor present. Hastily sketched notes will not be needed as a student can reproduce a class lecture by watching a saved sample provided by the instructor or by creating their own examples and experimenting with them. Ultimately it is our intention to provide animation building blocks that will allow instructors to create their own animations. However, the Working Group on Evaluating the Educational Impact of Visualization [16] reports that one reason instructors often do not use animations is that they do not have time to create their own. Hence we decided to initially concentrate on providing a set of our own animations with the tool.

The tool also currently includes a number of other features to aid an instructor or student. First, a property sheet allows an instructor to customize an animation for the textbook they are using or for their own teaching style. For example the pre-supplied captions can be edited and the names of the operations can be changed. This customization feature was added to address the concerns voiced by the visualization working group that animations were often not used because instructors could not customize the animations to the textbook they were using [16]. Second, the animations can be stepped through at the user's own pace and the animations can be reversed to return to a previous state.

The rest of this paper is organized as follows. Section 2 describes previous work. Sections 3-4 present an example interaction with the tool and a brief overview of its design and implementation. Finally the paper concludes with a discussion of potential future work and conclusions.

2. RELATED WORK

A great deal of work has been expended on creating various toolkits that allow instructors or students to either instrument code to produce an animation of an algorithm or else to write a script that produces an animation of an algorithm [15, 10, 26, 27, 4, 5, 22, 7, 8, 3, 23, 20, 1]. Our tool differs from these tools in that it produces data structures through sketching, not via a script or program. Another difference is that our tool provides captions that accompany the animation. One can also find animations of heaps on the web, such as [2]. However these animations tend to have more limited functionality than our tool, for example by hard-coding the values provided to the heap or only providing a limited set of operations, such as insert and extract max. There have also been studies that look at the effectiveness of animations. These studies imply that some type of active participation, rather than passive participation, is required to make the animations effective [24, 12, 6, 25, 18, 19, 21]. They also imply that well designed illustrations that contain both explanatory text and graphics can improve students' mastery of concepts, compared with purely textual instruction [13, 14, 17, 9, 11]. Our tool is meant to be used by instructors during interactive lecture sessions so while it is true that students can sit back and remain passive it is also true that students can actively participate and ask the professor to change values in the heap or change the structure of the heap and observe what happens as operations are performed. Additionally students can actively participate on their own by drawing their own example heaps and applying operations to these heaps. Finally we have incorporated a multi-view environment in our tool and plan to expand it in the future by allowing sound bites to be incorporated that describe the type of operation which is being performed.

3. AN EXAMPLE SESSION

In this section we will give a brief description of how a user can create and animate the operations on a MaxHeap. Figure 1 shows a sample screen shot of the tool. The tool has two modes, a construction mode for creating binary heaps and an operations mode for applying operations to the constructed heap.

A user can either sketch a binary heap or create one using the construction menu's fast build option. To sketch a tree the user starts by clicking anywhere on the canvas to create the initial root node. The user then clicks and drags from the root node to start creating the nodes of the binary heap. Anytime that the user clicks on a node and then drags the mouse away from the node a new child is created, provided that the node does not already have two children. To keep the creation of binary heaps simple we decided to support only one style of sketching. To determine which style of sketching was most "natural" for people, we asked a number of instructors and students to quickly draw binary trees. The majority of individuals drew a node, then drew edges from the node and created new nodes at the end of the edges. Hence it was the style we adopted.

Once the nodes have been created, or even while nodes are being created, the user can select a node and enter a value. Hitting the enter key will cause a left to right, top to bottom traversal of nodes so that all the nodes can be given values without moving the mouse. At any time the user can select the beautify button and the tool will draw a nicely formatted version of the tree.

The second way to create a binary heap is to use the quick structure builder menu. It asks for the number of levels in the tree and an optional number of leaves for the bottom level. The user can also ask that the heap be populated with a set of random values selected from a user specified range. Initially the heap property will not be satisfied. Figure 1 shows an example of a heap created using the quick build feature.

3.1 Max Heapify

When the user finishes construction of the tree, the user can select Build Max Heap to enter Operations Mode. The sketching tool will first verify that the heap is structurally sound, namely that it is a complete binary tree except for the bottom level, and that the leaves of the bottom level are filled from left to right. If the user has created an incomplete tree, the sketching tool will draw "ghost" nodes that make the heap structurally sound. The ghost nodes appear as gray nodes with gray edges. A dialog box also appears asking the user if the user wishes to accept the ghost nodes. Accepting the ghost nodes causes the sketching tool to introduce the nodes to the structure as full nodes. Rejecting the ghost nodes leaves the ghost nodes in place and allows the user to connect them individually. Canceling the operation removes the ghost nodes. Selecting either of the latter two options will prevent the heapify operation from taking place. Ghost nodes are intended to aid in the rapid building of trees for instructors and to aid students by providing a visual representation of the structural state required to heapify a binary tree. The heapify operation will also check to make sure that all nodes have a value and alert the user if one or more nodes is valueless.

Once the structure has been verified the sketching tool actually performs the heapify operation internally but does not yet reflect it on the screen. The user can press the Next button to start a step-by-step animation of the heapify process. The sketching tool brings the user's attention to the bottom right-most node in the tree that has children by flashing the children. Simultaneously a caption appears explaining the purpose of this step, which is to find the larger of the two children. When the user is ready to proceed to the next step the user again presses the Next button and the sketching tool flashes the larger of the two children and the parent node. Again a caption appears explaining that heapify is comparing the two nodes to determine which one is the largest. A third press of the Next button will cause the two nodes to be swapped if the child is larger than the parent (Figure 2). If the two nodes do not need to be swapped then the animation proceeds to the comparison of the next two nodes in the heapify process. Using Next, the user can step through every step in the heapify process. At any time, the user can use the Previous button to step backwards through the heapify process. Using Play will cause the animation of the heapify operation to occur continuously until it is either completed or the user presses the pause button.

3.2 Operations on the Heapified Tree

Once the structure has been heapified, the user can select operations from the operations menu to operate on the heap. Currently the tool supports inserting a new value into the heap, removing the maximum value from the heap, deleting a selected node from the heap, and updating the value of a node in the heap. As with the heapify operation, when the user selects an operation the sketching tool performs the operation internally and then allows the user to press the animation controls to control the animation of the operation.

3.3 Exiting Operations Mode

The user can return to sketching mode by either using the Exit Op Mode button or resetting the structure. Resetting the structure will return it to its original, un-heapified state. To return to Operations Mode, Build Max Heap must be used again. Repeating the heapify operation ensures that the heap is still structurally sound.

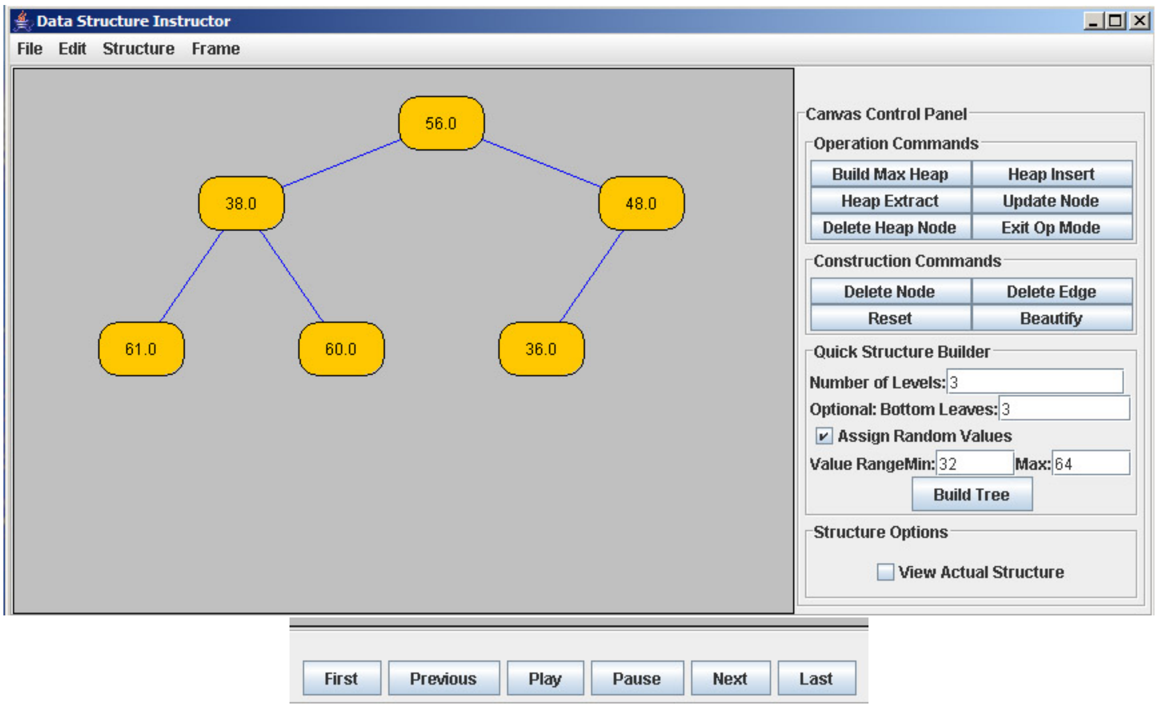
3.4 Editing Properties

A pop-up property sheet allows various aspects of a heap to be customized. For example, the names of the operations can be changed to match those found in the instructor's textbook. Thus an instructor could change the "Extract Heap" operation to read "deleteMax" instead. Similarly a property allows the heap to be treated as either a min or max heap.

4. DESIGN AND IMPLEMENTATION

The sketching tool is written in Java and is currently written as a stand alone application. Its class structure is designed to be easily extensible so as to allow new data structures to be rapidly added to the tool in the future. The class hierarchy used to build, modify, and animate a data structure are described below.

4.1 The Structure Class



Number of Levels: 3
Optional: Bottom Leaves: 3
✓ Assign Random Values
Value RangeMin: 32 Max: 64

Figure 1: A screenshot taken of a binary heap that has been created using the quick structure builder. As requested by the user, the binary heap contains three levels, with three leaves on the bottom level, and the heap has been populated with random values between 32 and 64.

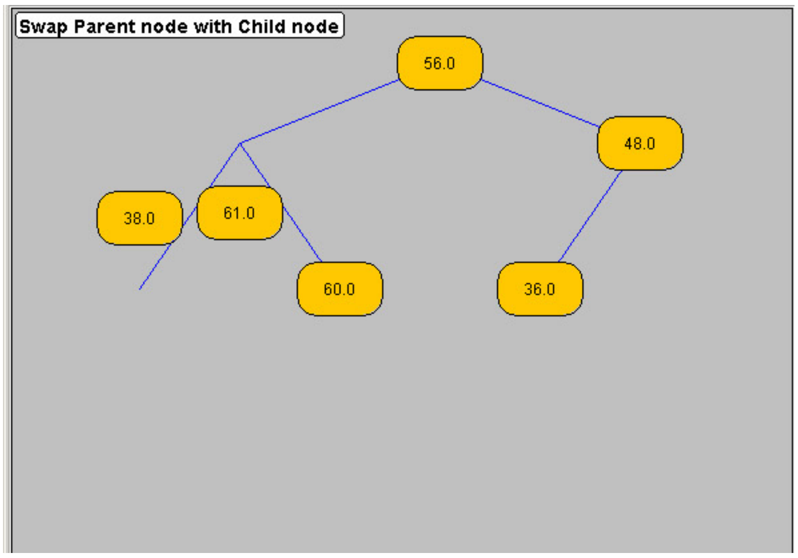


Figure 2: A screenshot taken while 61 is being promoted to a parent during the heapify operation.

The structure class is the abstract base class for all data structures used by the sketching tool. Our initial goal is to allow the rapid sketching of node and edge based structures, such as trees, lists and graphs, so the structure class declares methods for creating and deleting nodes, directing how nodes are connected with edges, and specifying ”wellformedness” rules for the structure. Additionally it holds the animation event queue, which will be described later. As an example, a binary tree extends the structure class by providing wellformedness rules that allow only two children for each parent and that establish a root node. Our Max-Heap data structure extends binary tree and provides functions to heapify, extract the max node, insert or delete nodes, and update the priority of existing nodes. Max-Heap's functions generate structure events and populate the structure's animation event queue. Max-Heap adds the additional wellformedness rule that the binary tree must be complete except for the bottom level.

4.2 The Structure Event Class

The structure event class provides a base class for animating objects in a data structure. Each class that extends the structure event class needs to provide methods for doing both an animation and reverse animation (i.e., restoring a data structure to its previous state). Additionally, an event class can provide an optional caption to be displayed when the event is executed. The caption should describe the sub-operation being performed (e.g., compare or swap). Operations such as heapify or insert-node generate the appropriate set of animation events and store them on the animation queue provided by the structure class. These events are then popped off or pushed onto the animation queue by the next, previous, and play buttons. Our max heap data structure makes extensive use of compare and swap events. These events can also be used by other data structures, such as a linked list data structure that is currently under development.

4.3 The Node and Edge Class

These abstract classes exist to implement the individual elements of the structure class and they include the unique attributes of the structure. For example, with a binary tree the classes are extended to binaryNode and binaryEdge respectively. binaryNode has a pointer to a parent and to two children. binaryEdge provides an unweighted edge that connects two nodes. Abstract draw methods available for node and edge allow their subclasses to draw each component and respond to any commands issued by the data structure.

5. FUTURE WORK

At the suggestion of one of our instructors we are currently adding an array at the bottom of the graphics window so that users can see how the array implementation of a heap looks. At a higher level we would ultimately like to allow instructors to be able to sketch a wide variety of data structures, including arrays, lists, trees, and graphs, and provide appropriate sets of operations for manipulating these data structures. We would also like to expose the lower level animation events, such as swap and compare, so that instructors can assign homework problems that students can solve by producing their own animations. Finally we would like to provide an area for displaying pseudo-code or source code and highlighting the statements in the code that would be executed to produce the effects being shown in the graphics window. Finally we plan to start using this tool in the classroom to gain feedback from students and instructors as to its effectiveness and usability.

6. CONCLUSIONS

The sketching tool described in this paper provides an interactive, fast, and educational way of both presenting and learning about binary heaps. It can be used in lectures, individual discussions with students, or by students themselves to quickly create custom heaps and to experiment with the effects that operations have on these heaps. The use of captions and animations allow the students or instructors to move through the operations in a step-by-step fashion with an explanation provided at each step as to what is happening. It is hoped that the use of this tool in the classroom can alleviate the difficulties that students often have in taking notes when instructors are rapidly changing diagrams since the students will be able to experiment with the diagrams themselves and at their own pace once class is over.

7. REFERENCES

[1] AKINGBADE, A., FINLEY, T., JACKSON, D., PATEL, P., AND RODGER, S. Jawaa: Easy web-based animation from cs 0 to advanced cs courses. In *SIGCSE Symposium on Computer Science Education* (Reno, NV, Feb 2003), ACM, pp. 162–166.

[2] ANG, W. Priority queue animation. 1998.

[3] BAKER, R. S., BOILEN, M., GOODRICH, M. T., TAMASSIA, R., AND STIBEL, B. A. Testers and visualizers for teaching data structures. In *ACM Technical Symposium on Computer Science Education* (New Orleans, LA, 1999), Proceedings SIGCSE'1999, pp. 261–265.

[4] BROWN, M. H. Exploring algorithms using Balsa-II. *IEEE Computer 21*, 5 (May 1988), 14–36.

[5] BROWN, M. H. Zeus: A system for algorithm animation and multi-view editing. In *Proceedings of the IEEE Symposium on Visual Languages* (Kobe, Japan, Oct 1991), IEEE Computer Society, pp. 4–9.

[6] BYRNE, M., CATRAMBONE, R., AND STASKO, J. Evaluating animations as student aids in learning computer algorithms. *Computers and Education 33* (1999), 253–278.

[7] DERSHEM, H. L., MCFALL, R. L., AND UTI, N. Animation of java linked lists. In *ACM Technical Symposium on Computer Science Education* (Cincinnati, Kentucky, 2002), Proceedings SIGCSE'2002, pp. 53–57.

[8] HAMILTON-TAYLOR, A. G., AND KRAEMER, E. Ska: supporting algorithm and data structure discussion. In *ACM Technical Symposium on Computer Science Education* (Cincinnati, KY, 2002), Proceedings SIGCSE'2002, pp. 58–62.

[9] HANSEN, S., NARAYANAN, N., AND HEGARTY, M. Designing educationally effective algorithm visualizations: Embedding analogies and animations in hypermedia. *Journal of Visual Languages and Computing 13*, 3 (2002), 291–317.

[10] HENRY, R., WHALEY, K., AND FORSTALL, B. The University of Washington illustrating compiler. *Sigplan Notices 25*, 6 (1990), 223–233.

[11] HUNDHAUSEN, C., DOUGLAS, S., AND STASKO, J. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing 13*, 3 (2002), 259–290.

[12] KEHOE, C., STASKO, J., AND TAYLOR, A. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies 54*, 2 (Feb 2001), 265–284.

[13] MAYER, R., AND ANDERSON, R. Animations need narrations: An experimental test of a dual-coding hypothesis. *Journal of Educational Psychology 83*, 4 (1991), 484–490.

[14] MAYER, R., AND ANDERSON, R. The instructive animation: Helping students build connections between words and pictures in multimedia learning. *Journal of Educational Psychology 84*, 4 (1992), 444–452.

[15] MUKHERJEA, S., AND STASKO, J. T. Toward visual debugging: Integrating algorithm animation capabilities within a source-level debugger. *ACM Transactions on Computer Human Interaction 1*, 2 (June 1994), 161–213.

[16] NAPS, T., COOPER, S., KOLDEHOFE, B., LESKA, C., ROSSLING, G., DANN, W., KORHONEN, A., MALMI, L., RANTAKOKKO, J., ROSS, R., ANDERSON, J., FLEISCHER, R., KUITTINEN, M., AND MCNALLY, M. Evaluating the educational impact of education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (Thessaloniki, Greece, 2003), ITiCSE, pp. 124–136.

[17] NARAYANAN, N., AND HEGARTY, M. Multimedia design for communication of dynamic information. *International Journal of Human-Computer Studies 57*, 4 (2002), 279–315.

[18] PALMITER, S., AND ELKERTON, J. An evaluation of animated demonstrations for learning computer-based tasks. In *Human Factors in Computing Systems* (New Orleans, LA, Apr 1991), Proceedings SIGCHI'91, pp. 257–263.

[19] PANE, J., CORBETT, A., AND JOHN, B. Assessing dynamics in computer-based instruction. In *Human Factors in Computing Systems* (Vancouver, Canada, Apr 1996), Proceedings SIGCHI'96, pp. 197–204.

[20] PIERSON, W., AND RODGER, S. Web-based animation of data structures using jawa. In *SIGCSE Symposium on Computer Science Education* (Atlanta, GA, Feb 1998), ACM, pp. 267–271.

[21] RIEBER, L., BOYCE, M., AND ASSAD, C. The effects of computer animation on adult learning and retrieval tasks. *Journal of Computer-Based Instruction 17* (1990), 46–52y.

[22] ROBLING, G., AND FREISLEBEN, B. Animalscript: an extensible scripting language for algorithm animation. In *ACM Technical Symposium on Computer Science Education* (Charlotte, NC, 2001), Proceedings SIGCSE'2001, pp. 70–74.

[23] STASKO, J. Using student-built algorithm animations as learning aids. In *ACM Technical Symposium on Computer Science Education* (San Jose, CA, Feb 1997), Proceedings SIGCSE'1997, pp. 25–29.

[24] STASKO, J. *Empirically Assessing Algorithm Animations as Learning Aids*. MIT Press, 1998, ch. 28, pp. 419–438.

[25] STASKO, J., BADRE, A., AND LEWIS, C. Do algorithm animations assist learning? an emporical study and analysis. In *Human Factors in Computing Systems* (Amsterdam, Netherlands, Apr 1993), Proceedings INTERCHI'93, pp. 61–66.

[26] STASKO, J. T. Tango: A framework and system for algorithm animation. *IEEE Computer 23*, 9 (September 1990), 27–39.

[27] STASKO, J. T. Using direct manipulation to build algorithm animations by demonstration. In *Human Factors in Computing Systems* (New Orleans, LA, Apr 1991), Proceedings SIGCHI'91, pp. 307–314.