The Proceedings of the

First Australasian Conference on Computer Science Education

Sponsored by the

ACM Special Interest Group on Computer Science Education

University of Sydney, Australia
July 3-5, 1996

Symposium Chair: Alan Fekete
University of Sydney

Program Chair: John Rosenberg
University of Sydney

Edited by: John Rosenberg
University of Sydney

Data Structures with Ada Packages, Laboratories, and Animations

Herbert L. Dershem, Hope College, Holland, MI, dershem@cs.hope.edu
Wendy L. Barth, University of Illinois, Champaign-Urbana, IL, barth@cs.uiuc.edu
Cheri J. Bowsher, Indianapolis Life Insurance Company, IN
Darrick P. Brown, Hope College, Holland, MI, dbrown@cs.hope.edu

Introduction

The data structures course is one of the oldest and most stable courses in the computer science curriculum. It has been present in all model curricula and curriculum recommendations from 1967 on, and its content has remained remarkably stable.

Over the history of the data structures course, many tools and approaches have been introduced and effectively employed. This paper describes a course that was designed using a combination of three such tools: the Ada programming language, algorithm visualization and animation, and laboratories with experimental algorithm analysis. The tools developed and used are described in detail.

The Ada Programming Language

The use of Ada in the data structures course was pioneered by Feldman [3] and more recently advocated by Silver [6]. Several very good data structures textbooks are based on the Ada language including Feldman [2], Weiss [9], Hillam [4], and Stubbs and Webre [8].

The advantages of Ada in a data structures course include the following:

- Packages and private types allow the complete implementation of abstract data types including encapsulation and the separation of specification from implementation.
- Generics enable students to work at a higher level of abstraction when constructing abstract data types.
- Exception handling can be included within abstract data types to further enhance encapsulation.

In the course described here, students were provided with a library of Ada packages which they used in their programming projects and laboratory exercises. This enabled the students to use the data structures in their own programs without needing to implement them in detail. The code from the packages was available for students to examine and was used in the class to aid in the understanding of data structure and algorithm implementation.

Packages that were provided in the library are:

- AVL Trees
- Rational numbers
- Unlimited precision integers
- Binary search trees
- Binary heaps
- Leftist heaps
- Linked lists
- Queues
- Stacks
- B-trees
- Splay trees

Some of these packages were adapted from those found in Weiss [9].

Algorithm Visualizations and Animations

Algorithm visualization and animation has been used successfully in data structures courses for some time. Examples are found in Brown [1] and Naps [5]. Tools have been described which facilitate the development of these animations. The tool chosen for use in the present project is XTango [7].

In the present data structures course, visualizations and animations are used for both classroom demonstration and laboratory exercises. The animations are intended to enhance student understanding of algorithms, particularly since the students do not write code to implement the algorithms in most cases. Many animations illustrate the algorithm through an animation that is viewed simultaneously with the Ada code which implements the algorithm. Ada statements in the code display are highlighted as their action is animated.

Many animations are provided with the distribution of XTango. Some of these were found to be appropriate for use in the data structures course, often with minor modifications. In addition, other animations were developed as a part of the course development project. Those developed were:

- Linked lists with insertion, deletion, and search

- Infix to postfix expression conversion
- Binary search tree insertion and deletion
- AVL tree rotation
- AVL tree insertion
- Splay tree rotation
- B-tree insertion

Example Animations

AdaVision is the name given to the algorithm animation system developed for this project. Adavision combines Ada code with dynamic images to serve as a teaching tool for data structure courses taught in Ada. Using the algorithm animation package XTango, animations are created so students may view the connection between Ada code and the action of algorithms on data and data structures. The Ada code associated with each algorithm appears in the display area of XTango. In some cases, procedures which are not explicitly displayed are used in order to simplify the code.

The structures and data involved in an algorithm are represented by images. These images move as the result of interesting events, such as the insertion of a node into a tree or the movement of a link in a rotation. At the beginning of each animation, the first line of Ada code is highlighted by a rectangular image. Succeeding lines of code are highlighted as they are executed. The user observes the image movement taking place in conjunction with the code highlighting. Two of the animations, Linked List and AVL Tree Rotations, are described below

Linked List

The list animation demonstrates how inserts, deletes, and finds are done on a linked list with a dummy header node. Insert may be done at any point within the list, delete will remove all occurrences of a particular value from the list, and find will search for the first occurrence of a value in the list.

XTango's animation window appears, and after the 'run animation' button is clicked, all interaction with the user occurs in the shell window. A menu is displayed there, giving the user the options of inserting a node, deleting a node, finding a node, or quitting the application.

If the user chooses to insert a node, s/he will be prompted for the value to be inserted, and then prompted for the desired place to insert the node: either at the start of the list, the end of the list, or after another user-specified node. If the user desires to delete a node, s/he will be prompted for the value to be deleted, and informed that all nodes containing the value will be deleted. If the user chooses to find a value in the list, s/he will be prompted for the value to find, and informed that only the first occurrence of the value will be found. After all information for a particular operation has been gathered from the user, the animation begins.

In an insert, an external pointer finds the node to be inserted after, and a new node is drawn and added to the list. In a delete, an external pointer finds both the node to be deleted and the node immediately before it prior to deleting the node. In a find, a comparison is animated between each element of the list and the find value. The find value appears in the lower left corner of the display area, and as each element is visited, the find value moves next to the value of the node. If the values match, the images flash. If they do not match, the find value returns to its place in the corner.

Nodes are represented by divided rectangles. The left half of the rectangle contains the value of the node, while the right half holds a pointer to the next node. Any external pointers, such as those used to find a certain node in the list, appear and move along the bottom of the list image. The code corresponding to each operation appears at the top of the animation window, and after an operation is completed, it is erased. The original list consists of a pointer named HEAD that points to a dummy header, that is, an empty node whose pointer field points to NULL. As lists become long, they will move off the display area to the right. The images can still be viewed by using the arrow buttons on the left side of the XTango window

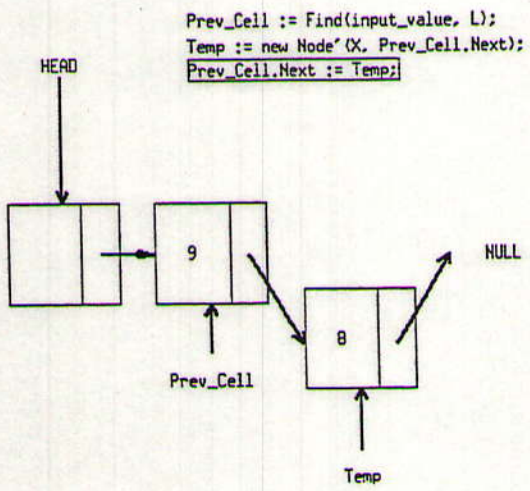

Figure 1. Inserting a node at the start of a linked list

AVL Tree Rotation

AdaVision's 'avlrotat' demonstrates single and double rotations of an AVL tree upon the insertion of an element. The user may view any of four rotations as many times as desired by choosing an option number in a shell window after clicking XTango's 'Run Animation' button.

In the single left rotation, there exist nodes A and B, where A is the original root of the tree and B, A's left child, is the root which results from the rotation. The tree is filled in by three triangles which represent subtrees of depth 'n'. The

double left rotation consists of three node images: A, the original root; B, A's left child; and C, which is B's right child and the new root. In this case, the tree is filled by two triangular subtrees of depth n, and two triangular subtrees of depth 'n+1'. The nodes in each rotation are connected to subtrees by way of 'links,' which serve as pointers to nodes in the tree. The single right and double right rotations are mirror images of the left rotations.

The element is represented by a small, orange triangle which first appears in the top right-hand corner of the XTango window. After working its way down the tree in standard binary search tree fashion, the element attaches, or inserts, itself to the bottom of a subtree, potentially causing the tree to become unbalanced. The element is inserted into the left-most subtree of the pivot for the single left rotation, the right-most subtree for the single right rotation, the right subtree of the left child of the pivot for the double left rotation, and the left subtree of the right child of the pivot for the double right rotation.

Each rotation also has its own display of Ada code. As each line of code is highlighted, the appropriate link movement is performed. Once all links are in position, the rotation occurs. The image at each node moves to its new position in the balanced tree and the links are redrawn accordingly.

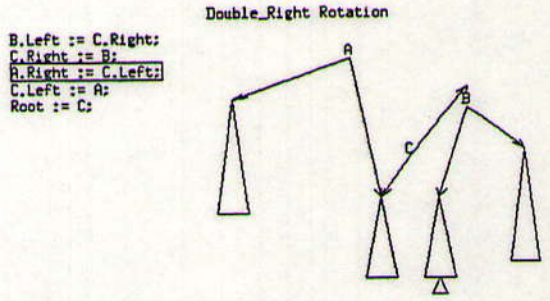

Figure 2. The repositioning of links due to the insertion of an element into a subtree.

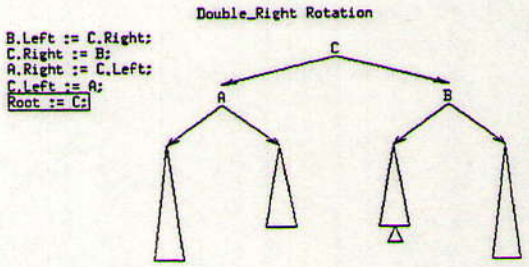

Figure 3. A balanced AVL tree following the performance of a double right rotation.

The Laboratories

There are eight laboratory exercises written for this course. Some of these require the students to use a package called THREADS (Test Harness for the Repeated Execution of Ada on Data Structures). THREADS is described in the following section.

Titles and brief descriptions of the laboratories are given below.

1. Writing an Ada program
   The students are introduced to the Ada language by writing a program to compute the nth power of 2 using integers and floats. They are also required to write a program which uses Newton's method to calculate the square root of 2. Students observe the limitations of size and accuracy with Ada's built-in numeric types.
2. Using Ada Packages
   Students use a package called Big_Integer to obtain results for larger powers of 2. They also use a rational number package to obtain more accurate results for the square root of 2.
3. Using Generic Packages
   Students use a generic rational package and instantiate it for Big_Integer to increase the accuracy of the square root of 2 calculation.
4. Big Oh Sampling
   Five algorithms are provided in Ada programs with various Big Oh values. Students run these through THREADS to observe their timing behavior both in tabular and graphical form.
5. Big Oh Determination
   Students work with 10 algorithms whose big Oh behaviors they must analyze and observe.
6. Stacks and Queues
   Students run animations in XTango to observe and analyze the behavior of a stack (Infix to Postfix conversion) and a queue (Post Office Queue Simulation).
7. Comparison of AVL and Binary Search Trees
   Students use packages for AVL trees and Binary Search trees to observe and compare their behaviors in terms of search/insertion times and average depth of an element in the tree. THREADS is used to perform the analysis on the observations.
8. Sort Comparisons
   THREADS is used to compare the behavior of five different sort algorithms over various data distributions.

THREADS: A Data Structures Laboratory Test Harness

Many experiments that are performed in the laboratories involve running tests on algorithms that have been implemented using Ada packages. These tests produced results

that can be measured and analyzed. Working in the lab gives students the chance to be more directly involved in their learning, increasing the amount of information they retain.

Some of the Ada packages will be written by the students themselves, but more are provided by the instructor. In this way, the students are exposed to more data structures and algorithms. Students will spend their time seeing and experiencing the effects of algorithms instead of actually coding the algorithms and corresponding data structures. This should increase their ability to analyze the effectiveness and/or efficiency of different approaches to a problem.

THREADS (Test Harness for Repetitive Experiments on Ada Data Structures) is a tool that can be used to run tests on data structures and algorithms, reporting back to the user some type of the measurement of the test. The tests are 'black box' programs that are implemented separately, and may be tested and run separately as well. An abbreviated manual for THREADS follows.

The basic idea behind THREADS is illustrated by the following chart

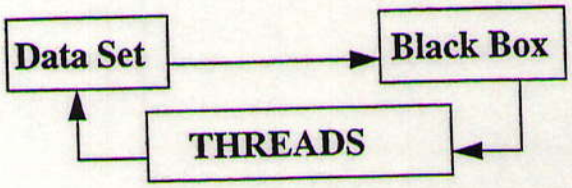

Figure 4. Configurations for THREADS

THREADS generates a data set based on information given by the user. This data set is used by a black box to run one experiment. Upon the black box's completion, it returns to THREADS the sample size of the data set and an integer measurement of the test. The measurement will be included in a table that keeps track of each experiment the user runs.

Running THREADS brings up the interface shown in Figure 5. All information needed for the data set is input in the appropriate places by the user. The parameters the user may designate are as follows:

Method: The black box to use for the experiment

Write to File: The named file where the data set is stored. If no file is designated, a

temporary default file will be used.

Write Path: The path to the directory where all data and files will be written.

Use File: The path and name of a data set to be used in place of a file generated by THREADS.

Sample Size: The number of elements in the data set.

Sample Distribution: The statistical probability distribution used to generated the random data set.

Sample Order: The extent of ordering imposed on elements in the data set. The default settings are for a 100 element, completely unsorted data set generated randomly from a uniform random distribution.

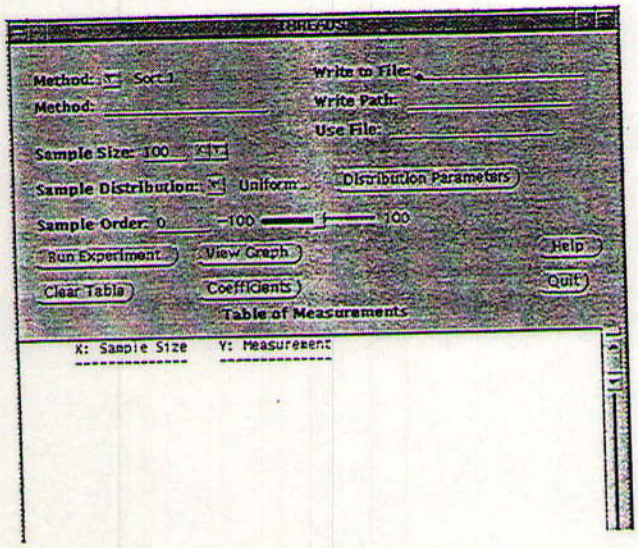

Figure 5. THREADS Window

Method: The black box process is spawned by the THREADS process. When THREADS executes a black box, it gives the black box a data set generated by THREADS. It then waits for the black box to return. When the black box returns, THREADS takes the data and writes it to the Table of Measurements. The black box returns 2 integers. The first is the size of the data set and the second is the measurement that the black box returns.

The meaning of the measurement returned by the black box will vary depending on which black box is being run. In some cases the measurement may be the number of comparisons that were performed in a sort routine. In the case of the binary search tree experiments, the measurement represents the average depth of a node in the tree. In all cases, however, the measurement will be a non-negative integer useful in analyzing the effectiveness or efficiency of a certain data structure or algorithm for a particular data set. The measurements returned from different experiments can then be compared against each other to aid the user's analysis.

Write to File: The text field labeled "Write to File:" takes a name as input. If a name is specified, the generated data set will be saved to a file with that name in the directory specified in the "Write Path:" text field. If no name is specified, the data set will be saved to a temporary file.

Write Path: The "Write Path:" text field takes a path string as input. THREADS will not operate until a valid path is

given. The path string needs to be a path where the user has read and write permissions. THREADS reads and writes many data files. If it cannot read and write its data, it will not work properly. If the user attempts to run a black box without suppling the write path, a notice prompt will appear and notify the user to supply THREADS with the appropriate information.

Use File: The "Use File:" text field takes a string as input. This text string must contain the entire path and name of the data file to be used. If a valid path and name is given, THREADS will use this data set for the black box instead of generating a new data set. THREADS will use a specified data set before generating a new data set. Therefore, if the user wishes to generate a new data set, the string in the "Use File" text field must be deleted.

Sample Size: The sample size field allows the user to enter the number of elements to be included in the data set, ranging from 1 to 10000. The sample size may be changed by using the mouse to click on the up-down arrows, or by manually entering the size into the text field. The default is 100 elements.

Sample Distribution: Sample distribution indicates the type of randomness in which the data elements are to be distributed. There are six different distributions to choose from: uniform, exponential, normal, gamma, Poisson, and binomial.

To the right of the Sample Distribution, there is a button labeled 'Distribution Parameters'. If this button is clicked a window panel with number fields will appear.

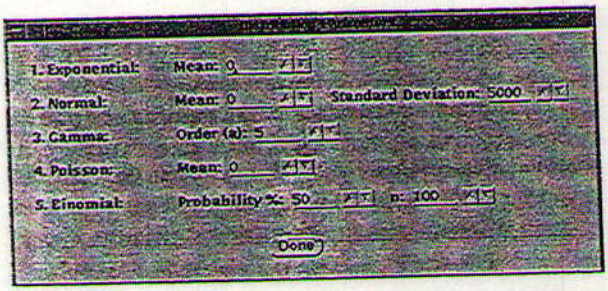

Figure 6. Distribution Parameters Window

With this distribution window panel, the user can modify the distributions by changing the parameters for each distribution.

These distributions can be used to evaluate how the distribution of data can affect different data structures. For most cases, Uniform distribution is sufficient. Future work on distributions includes the development of black boxes that fully utilize the Sample Distribution feature of THREADS.

Sample Order: The sample order refers to the degree of order the user would like in the data set to be generated, ranging from -100 to 100. A sample order of 100 means that 100% of the data will be in increasing sorted order. A sample order of -100 means that 100% of the data is in decreasing sorted order. A sample order of zero means that the data is in perfectly random order. Any value between -100 and 100 is acceptable. A value of 50 means that the first 50% of the data is in increasing sorted order, the remainder is in random order.

Data Set: The data set is generated based on the information from the sample size, distribution, and order fields. The elements are randomly generated to fulfill the user's requirements. A data set can also come from a imported data set using the "Use File:" text field by supplying a path and name.

Table of Measurements: Since data sets may be saved in files designated by the user, experiments may be repeated. The table of measurements from an experiment session may also be saved, so the user may come back to the data at a later time to continue analysis or even add to the previous experiment record. Tables are saved by clicking the right mouse button while on the table. From the 'File' menu, choose the option 'Save as...' and a save window will appear. To load in a previously saved table, choose the option 'Open' from the 'File' menu.

Run Experiment: When the 'Run Experiment' button is clicked, the data set is generated and written to the appropriate file. Next, the black box process is spawned and executed. When the black box finishes, the sample size and measurement are written to the table of measurements. If the user has not provided THREADS with the appropriate information, the user will be notified to do so and no experiment will be run. If the black box aborts or crashes, the user will be notified that there was an error in the black box and no data will be written to the table.

View Graph: When the 'View Graph' button is clicked, the measurements currently in the table will be used as the coordinates for a graph. Graphs are generated using the X-Windows graphing package XVGR and may be created at any point in the experiment session. Each graph is produced in its own window with a unique title, which allows for easy comparison between graphs.

Clear Table: The 'Clear Table' button allows the user to clear the table at any point during a THREADS session. This enables the user to start a new set of experiments at any time. When the 'Clear Table' button is clicked a prompt will appear asking if they really want to clear the table. If "Clear Table" is selected, the table will be cleared. If cancel is selected, the user will be returned to THREADS with no changes.

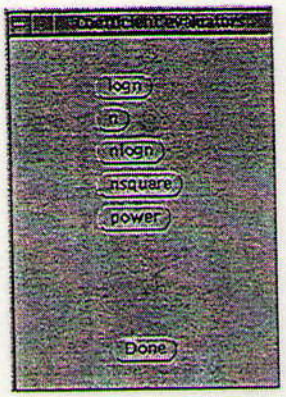

Figure 7. Exponent selection for Big Oh coefficient summation

Coefficients: If the 'Coefficients' button is clicked, a small window with five buttons will appear (Figure 7). The five buttons are log(n), n, nlog(n), n2, and n to some power, 'n' being the sample size. If one of these buttons is clicked, an window will appear displaying the coefficients of that particular Big Oh of the data in the table. For example, if the data in the table is:

100 600

If the n^2 button is clicked, the window will appear displaying:

100 600 6.000000000E-02

This means that with n=100 and y=600, an expression of the form $y=cn^2$ would require c to be 6.0000000E-02. If this coefficient remains relatively constant over many values of n, the function represented is a good candidate for the big-oh function of the black box process.

Help: Help may be found both by clicking the 'Help' button on the THREADS window, or by pressing the help key on the keyboard. The button on the THREADS window will open a pop-up window that contains complete help text. Pushing the 'help' key on the keyboard will give a short summary of help for the spot on the window where the cursor is pointing.

Quit: If the 'Quit' button is clicked, a notice prompt will appear and ask if the user really wants to quit. If cancel is selected, the user will be returned to the main THREADS program without any changes. If 'Quit' is selected, THREADS will exit and close all windows. Also when THREADS is quit, all temporary files will be deleted so no unwanted files remain in the specified write path directory.

THREADS Tutorial: There is a small tutorial program that is included with THREADS. When this program is run a window opens that displays the complete tutorial text in a scrollable area. This window is sized so that it can be placed next to the THREADS window on the same screen for easy reference when working with THREADS.

Conclusions

The student reactions to the use of the laboratories was favorable. They found THREADS to be a very useful tool as well as easy to use. Their evaluation of the laboratories was uniformly positive, indicating that the students found them interesting, instructive, and challenging.

This course was offered in a setting where the laboratories had to be completed outside of regularly scheduled class time and without the instructor's supervision. It would have been better to have scheduled and supervised laboratory sessions for completion of the laboratory exercises. Also, the setting used made the laboratory exercises individual efforts whereas a team effort would be more effective.

The animations were used in two settings. The students were given instructions on their use and encouraged to run and observe them. In addition, they were used for in-class demonstrations. A closed laboratory setting would increase the effectiveness of the animations by permitting an effective combination of demonstration and interactive student use.

The success of the substitution of animations for student-written implementations is best measured by the ability of the students to understand the data structures and associated algorithms and to apply them in their later work. Since all students from this class have now completed a subsequent algorithms class, we can state that students learning in this environment were able to understand, apply, and extend the concepts of this course as well as those who did more extensive implementation programming.

The most problematic aspect of these modifications was the use of the Ada programming language. The student reaction to the use of Ada was uniformly negative. This was not a result of the language itself, which the authors still believe to be a very effective vehicle for the demonstration of data structures, but rather due to local conditions.

In all courses in the curriculum prior to this course, the students had used Pascal. In courses beyond this one, they are either using C++ or given a choice of language. The only other course in the local curriculum that uses Ada heavily is the programming languages course. As a result, the students found the on-the-fly learning of Ada at the beginning of the data structures course to be an unnecessary burden. They felt that this burden came on top of their other responsibilities for the class and that there was little future value to be derived from this effort.

As a result of the above observations, the future direction of this effort is to convert the data structures course, it labora-

tories, and animations to C++. The local curriculum has now been modified so that the students first program in C++ in a course prior to this one, so there will be no time required to learn the language. In addition, the previous courses now emphasize the object-oriented approach, and this approach will be integrated into the data structures course and the laboratories as well.

The introduction of closed laboratories is another modification that will be instituted in future offerings of this course.

Acknowledgments

Work on this project was supported in part by DARPA grant number MDA972-92-J-1030 and National Science Foundation grant number CDA-9200118.

BIBLIOGRAPHY

[1] Brown, M.H., *Algorithm Animation*, Cambridge, MA, MIT Press, 1987

[2] Feldman, M.B., *Data Abstraction with Ada*, Reston, VA, Reston Publishing Company, 1985/

[3] Feldman, M.B., Teaching data structures with Ada: an eight year perspective, *SIGCSE Bulletin*, 22(2):21-29, June, 1990.

[4] Hillam, B., *Introduction to Abstract Data Types Using Ada*, Englewood Cliffs, NJ, Prentice-Hall, 1994

[5] Naps, T.L. Algorithm visualization in computer science laboratories, *SIGCSE Bulletin*, 22(1):105-110, February, 1990.

[6] Silver, J.L., Using Ada to specify and evaluate projects in a data structures course, *SIGCSE Bulletin*, 23(1):337-340, March, 1991.

[7] Stasko, John T. TANGO: A framework and system for algorithm animation, *Computer*, 23(5): 27-38, 1990.

[8] Stubbs, D.F. and N.W. Webre, *Data Structures with Abstract Data Types and Ada*, Boston, PWS Kent, 1993.

[9] Weiss, Mark A. *Data Structures and Algorithm Analysis in Ada*, New York, Benjamin Cummings Publishing Co. Inc., 1993.