



DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714



January 16, 1992

Herbert L. Dershem
Department of Computer Science
Hope College
Holland, MI 49423

Subject : BAA Proposal
Use of Ada, Laboratories, and Visualization in the Teaching of Data Structures
Log# 25
SISTO, BAA 91-18

Dear Mr. Dershem:

This letter is in reference to your proposal submitted in response to the *Commerce Business Daily* issue of 16 July 1991 for a DARPA-sponsored program of Curriculum Development in Software Engineering and Ada, DARPA/CMO BAA 91-18.

Your proposal was evaluated in accordance with the criteria set forth in the announcement, and is one of several determined to be a potential candidate for funding. Since it is not anticipated that available funds will be sufficient to support all such proposals, we are unable to be more specific at the present time, except to say that should your proposal then be selected for funding, you will be contacted by our Contracts Management Office.

Thank you for your participation in this procurement. Your efforts in expressing the ideas and concepts of your proposal are appreciated. We anticipate the publication of a new, Ada Undergraduate Curriculum Broad Agency Announcement by the end of February. DARPA wishes to encourage your participation in this and all future programs

Sincerely,

Dr. John F. Kramer
Program Manager
Software and Intelligent Systems Technology Office

cc: James M. Gentile

A. Cover Page

Broad Agency Announcement 91-18

CURRICULUM DEVELOPMENT IN SOFTWARE ENGINEERING AND ADA

Category 1 Proposal

Proposal Title: Use of Ada, Laboratories, and Visualization in the Teaching of Data Structures and Discrete Mathematics

Technical Point of Contact: Herbert L. Dershem
Department of Computer Science
Hope College
Holland, MI 49423
(616) 394-7508
dershem@cs.hope.edu

Administrative Point of Contact: James M. Gentile
Dean for the Natural Sciences
Hope College
Holland, MI 49423
(616) 394-7714



HOPE COLLEGE

DEAN FOR NATURAL SCIENCES

September 23, 1991

BAA #91-18
DARPA/SISTO
3701 North Fairfax Drive
Arlington, VA 22203-1714

TO WHOM IT MAY CONCERN:

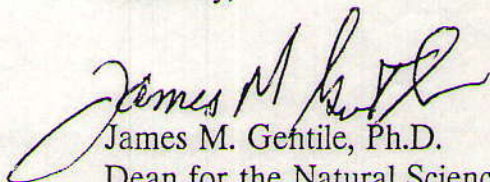
It is with significant enthusiasm and excitement that I endorse the ideas and concepts presented by the Department of Computer Science at Hope College in this proposal entitled "Use of Ada, Laboratories, and Visualization in the Teaching of Data Structures and Discrete Mathematics."

The program described in this proposal is timely and innovative. It will allow students to more easily bridge perceived disciplinary gaps and it establishes a firm foundation of connectedness for students studying mathematics and computer science. The property of connectedness in science and mathematics is crucial because it provides the learner with a tangible means of knowing. The proposed curricular endeavor is welcoming of students and it is understanding of the variances in backgrounds that students bring with them from high school. It is designed to meet students "where they are" and help students to achieve their fullest potential of understanding.

Hope College is committed to support this program in every way. My office will work with Professor Dershem in a coordinated fashion to insure programmatic success.

We are thankful for the opportunity to submit this proposal and we look forward to developing a vital program that will provide students with the tools necessary for future success in computer science.

Sincerely,



James M. Gentile, Ph.D.
Dean for the Natural Sciences and
The Kenneth G. Herrick Professor of Biology

JMG:b

B. Description of the Project

Summary of Project

The proposed project is for the development of a new course in Data Structures that includes the following features:

1. The use of Ada to aid in the development of the concept of Abstract Data Types.
2. A laboratory component where students would meet in a laboratory setting once a week to carry out experiments in data structures.
3. The use of algorithm visualization and animation techniques for both classroom demonstration and laboratory activities.
4. The integration of discrete mathematics topics into the course, including graph theory and recurrence relations.

Current Situation

Hope College is a four-year liberal arts institution with enrollment of approximately 2,500. The college has had a computer science department since 1974. The department presently consists of four full-time faculty members, three of whom hold Ph.D. degrees in Computer Science. The department graduated 14 majors in 1991 and will graduate a similar number in 1992.

The data structures course at Hope College is closely based on the CS7 course of ACM Curriculum 78 [1] and CO2 course of the ACM Liberal Arts Curriculum [11]. It is taken by Computer Science majors in the sophomore year, with Introduction to Computer Science and Computer Science II as prerequisite courses. Since 1985, data structures has been taught with Modula 2 as the primary programming language. Prior to that Pascal was used. Beginning in 1990, the computer science department began a curricular program that introduces the topics commonly included in a discrete structures course into the first four courses of the computer science curriculum, one of which is the data structures course. The discrete structures course, which was taught within the mathematics department, was consequently eliminated as a requirement for the computer science major.

Proposed Activities

The proposed project would seek to develop materials to support a course in data structures that would use Ada as the implementation language, integrate discrete mathematics topics into the course, and provide laboratories that would engage the students in experimental work and algorithm visualization. The Project Director would produce, as a result of this project, course materials including syllabus, a lab manual, Ada software used in support of the laboratory, and

handouts supporting the discrete mathematics component of the course.

It is proposed that the course and laboratory design be completed and supporting software written during the summer of 1992, that the course be taught and written laboratory exercises be produced during the Spring Semester of 1993, and that the finished lab manual and a paper describing the results of this curricular development be written during the summer of 1993.

Technical Approach and Rationale

1. Introduction of Ada as the primary language in a Data Structures course.

The use of Ada in the data structures course has been pioneered by Feldman [9] and more recently advocated by Silver [13]. The advantages of Ada as outlined in [10] include the following:

1. Data abstraction is enhanced by the ability to return any structure as the result of a function.
2. Packages allow for the implementation of ADTs along with the separation of specification from implementation.
3. Private types enhance the ability to implement encapsulation.
4. Generics enable the students to work at an higher level of abstraction in the construction of ADTs.
5. Exception handling can be included as an integral part of the ADT.

In addition, an early exposure to Ada in the Computer Science curriculum makes that language available as a tool in all later courses. In particular, Ada is already used as a primary language in the programming languages and concurrent systems courses, and its use in data structures would reduce the amount of time needed in these later courses to familiarize the students with Ada. Furthermore, students would be more likely to choose Ada in later courses where they have a choice of which language to use.

2. Introduction of a laboratory component into the data structures course.

A significant amount of attention has been given in recent computer science curriculum studies to the introduction of laboratory experiences [3] [14]. At Hope College, we have already begun efforts to introduce laboratories into the first two courses in the curriculum, that is, those that are prerequisites to the data structures course. Based on the success of those implementations and the recommendations of the ACM Curriculum committee, we propose the extension of the laboratory approach to the data structures course.

The present data structures course meets for three hours of lecture per week. The proposed course would meet for a two-hour laboratory in addition to the three hours of lecture. The number of credit hours for the course would be increased from three to four. In the laboratory, the students would be provided with a workstation and the activities would be specified in a written lab description and through verbal instructions from the instructor. The laboratory activities would include, but not be limited to, the following:

Implementing applications in Ada using provided ADTs.

Conducting experiments through the observation of timings of various algorithms to hypothesize or verify the run-time efficiency.

Observing the change in behavior in algorithms when different data structures or different implementations are used.

Analyzing the behavior of algorithms and data structures through the use of visualization and animation software.

Illustrating and emphasize mathematical concepts through the use of computer simulations.

3. The use of algorithm visualization and animation software.

We have already begun exploring the use of algorithm visualization and animation software in laboratories at Hope College in the Introduction to Computer Science course and in the Algorithms course. This technique has been used with great success by others as well [2] [12].

We propose integrating the use of this software to provide laboratory experiences and classroom demonstrations in the data structures course.

Because visualization reveals patterns in the changing distribution of an array of values as they are being sorted, it can aid students in doing in-depth analysis and comparison of sorting algorithms. Complicated schemes for implementing balanced search trees can be understood more clearly through graphics display and animation of the tree structures. Algorithms for searching and performing computations on graphs can also be illustrated at the conceptual and intuitive level through visualization techniques. For large data structures, graphical display of statistical information can lead to a better understanding of the asymptotic behavior of algorithms.

4. Inclusion of mathematics in the data structures course.

Discrete mathematics plays an important role in a data structures course. The approach we are implementing for providing computer science students with an adequate background in discrete mathematics is through the inclusion of the mathematics material in the first four courses in the

computer science sequence. This has several advantages over the offering of a separate course on discrete mathematics: (1) The mathematics that is needed in the computer science curriculum is covered prior to or at the same time as the computer science topic that needs it; (2) The students learn the mathematics in the context of its application in computer science, providing additional motivation for learning it; (3) The computer can be utilized as a tool to assist in learning the mathematics through computer simulation.

Previous Related Work

Professor Dershem has been active in computer science curriculum development for more than twenty years. His first activity was in the design of a course that combined the teaching of statistics and computer science [4]. His work on that project was supported by a grant from the National Science foundation and resulted in the publication of a laboratory manual for use in such a course [5].

Professor Dershem was also funded by NSF for the development of a modular approach to the teaching of introductory computer science [6]. As a part of this project, two modules on problem solving were produced [7] [8].

More recently, Professor Dershem has co-authored a programming languages text that uses Ada as the primary language for the discussion of imperative language concepts [10]. This text has been used at Hope College for the past four years in both its pre-publication and published forms.

Professor Dershem has also been active in curriculum development and in the activities of the Special Interest Group on Computer Science Education (SIGCSE) of the ACM. He served as program chair of the 1988 SIGCSE Symposium and edited the proceedings of that symposium [9].

Facilities

Three computer networks are in place on the Hope campus on which this project could be implemented. The choice of the platform for these activities will be made based on the status of these networks at the time of implementation. The three networks, which we will call the Sun, VAX, and PC networks, are described in the Appendix of this proposal. The college already has the Verdix VADS Ada environment available on the VAX network. If the Sun or the PC networks are used for implementation, then Hope College will purchase a suitable Ada environment for use on the chosen system.

If the VAX or PC networks are chosen for implementation, labs will meet in a classroom set up on campus that contains 28 Swan 386SX microcomputers that are on a local area network. These computers are arranged in a classroom setting suitable for conducting the lab described in this proposal. Each of these systems also has communications capabilities with the VAX network so that the use of either the PCs or the VAX can be implemented in this laboratory.

The Physics and Mathematics departments at Hope College recently received a grant from the NSF ILI program for the establishment of a laboratory of 20 X-terminals for the instruction of calculus and physics. The exact nature of this laboratory has not yet been determined, but it is possible that this laboratory, with the terminals connected to the Computer Science department's Sun network, could be used for the laboratory activities of this project.

The decision as to which facilities to use will be made after the specifications and availability of the X-terminal laboratory have been determined. Algorithm visualization and animation development software available on Hope College systems includes GAIGS on PC systems and TANGO on the Suns.

Bibliography

- [1] ACM Curriculum Committee on Computer Science, Curriculum '78: Recommendations for the undergraduate program in computer science, *CACM*, 22(3): 147-166, March 1979.
- [2] Brown, M.H., *Algorithm Animation*, Cambridge, MA, MIT Press, 1987.
- [3] Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A.B., Turner, A.J., and Young, P.R., Computing as a discipline, *CACM*, 32(1):9-23, January, 1989.
- [4] Dershem, H.L., A course on computing and statistics for social science, *Proceedings of 1972 Conference on Computers in the Undergraduate Curricula*, Atlanta, GA, 1972.
- [5] Dershem, H.L., *Computer Exercises for Elementary Statistics*, Wentworth, NH, Compress, Inc., 1979.
- [6] Dershem, H.L., A modular introductory computer science course, *SIGCSE Bulletin*, 13(1):177-181, February, 1981.
- [7] Dershem, H.L., *UMAP Module 477: Computer Problem Solving*, Cambridge, MA, Birkhauser Boston, Inc., 1981.
- [8] Dershem, H.L., *UMAP Module 478: Iteration and Computer Problem Solving*, Cambridge, MA, Birkhauser Boston, Inc., 1981.
- [9] Dershem, H.L. (ed.), *Proceedings of the Nineteenth SIGCSE Technical Symposium*, Association for Computing Machinery, 1988.
- [10] Dershem, H.L. and Jipping, M.J., *Programming Languages: Models and Structures*, Belmont, CA, Wadsworth Publishing Company, 1990.
- [11] Feldman, M.B., *Data Abstraction with Ada*, Reston, VA, Reston Publishing Company, 1985.
- [12] Feldman, M.B., Teaching data structures with Ada: an eight year perspective, *SIGCSE Bulletin*, 23(1):337-340, March, 1991.
- [13] Gibbs, N.E. and Tucker, A.B., Model curriculum for a liberal arts degree in computer science, *CACM*, 29(3):202-210, March, 1986.
- [14] Naps, T.L. Algorithm visualization in computer science laboratories, *SIGCSE Bulletin*, 22(1):105-110, February, 1990.
- [15] Silver, J.L., Using Ada to specify and evaluate projects in a data structures course, *SIGCSE Bulletin*, 23(1):337-340, March, 1991.
- [16] Tucker, A.B. et al (eds.), *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force*, New York, ACM Press, 1991.

Section C. Summary of Deliverables

1. Laboratory Manual for Use in Data Structures Course.

A laboratory manual suitable for use in conjunction with a data structures course will be developed as a part of this project. This manual will contain a description of at least 20 laboratory projects which can be carried out in a supervised lab setting, or alternatively, can be given as outside-of-class assignments. These laboratories will include developing applications using prepared Ada packages, experimental comparison of algorithms and data structures to observe and analyze run-time behavior, computer simulations to illustrate mathematical concepts, and algorithm visualizations and animations. This laboratory manual will be published in a form suitable for distribution to others interested in implementing a data structures laboratory and provided upon request for the cost of production.

2. Ada packages to support the Data Structures Laboratories.

A number of Ada packages will be constructed for use in the laboratories. These will include packages for stacks, queues, sequential lists, trees, and graphs. These packages will all be made available to any interested parties for use in data structures or other relevant courses.

3. Visualizations and Animations.

All data structure and algorithm visualizations and animations developed for laboratories and classroom demonstrations will be made available for use by others.

4. Paper for submission to *SIGCSE Bulletin*.

A description of the data structures course and the associated materials will be described in a paper submitted to the *SIGCSE Bulletin* and for presentation at the SIGCSE Technical Symposium.

D. Summary of Schedule and Milestones

All work on this project will be carried out by the Project Director with possible assistance from a student, contingent on the availability of local financial support for the student assistant.

Date	Activity
June 1992	Design data structures course and laboratories
July 1992	Write Ada packages, data structure visualizations, and algorithm visualizations/animations to support laboratories
Jan-Apr 1993	Project Director will teach the newly designed course and concurrently produce the laboratory manual
May 1993	Project Director will produce the final bound lab manual and write paper about project for SIGCSE

E. Proprietary Claims to Results

The Project Director makes no proprietary claims to any results or other artifacts supporting and necessary for the use of this course.

F. Qualifications of Project Director

CURRICULUM VITAE

Herbert L. Dershem

Education:

B.S. University of Dayton, 1965

M.S. (Computer Science) Purdue University, 1967

Ph.D. (Computer Science) Purdue University, 1969

Experience:

Hope College, Assistant Professor, 1969-74

Associate Professor, 1974-81

Professor, 1981-present

Chair of Computer Science Department, 1976-present

Oak Ridge National Laboratories, Visiting Research Scientist, 1977-78

Boston University Overseas Program, Visiting Professor, 1982-83

Honors and Awards:

NDEA Fellow, Purdue University, 1965-68

Honeywell Corporation Fellow, Purdue University, 1968-69

Project COMPUTe Awardee, Dartmouth College, 1972

NASA/ASEE Summer Fellow, Goddard Space Flight Center, 1976

Oak Ridge Associated Universities Summer Fellow, 1977

Grants:

Co-director, "Introduction of the Computer in the Statistics Curriculum", NSF Office of Computing Activities, 1971-73

Director, "A Modular Approach to the Introductory Course in Computer Science", NSF Local Course Improvement Program, 1978-80

Co-Director, "A Microcomputer Laboratory for use in Teaching Statistics", NSF Instructional Scientific Equipment Program, 1979-80

Director, "CSNET Membership in Support of Computer Science Research", NSF RUI Program, 1987-90

Publications: (23 total, those pertinent to this project listed in Bibliography)

Other major sources of support: None

Related proposals pending: None

G. Budget

June-July 1992	Project Director's Salary (2/9 academic year salary)	\$11,800
	Benefits for Project Director (30% of salary)	\$ 3,540
May 1993	Project Director's Salary (1/9 academic year salary)	\$ 5,900
	Benefits for Project Director (30% of salary)	\$ 1,770
Total Request		\$23,010

A Hope College contribution will be providing 1/3 release time for the Project Director during the Spring 1993 semester while he is teaching the course and writing the laboratory manual. The approximate amount of that contribution would be 1/6 of the Project Director's academic year salary which would be \$8,800 based on his 1991-92 salary. In addition, all hardware and software needed to implement this project will be provided by Hope College.

H. Appendix - Description of Hope College Computer Networks

Computer Science Department Sun Network

Machine/Part	Peripherals
Sun 4/360	32 MB memory, 688 MB disk, 2400 baud modem
Sun 4/470	32 MB memory, 669 MB disk
(2) Sun 4/40s	12/16 MB memory, 207 MB disk, 3.5" floppy
(8) Sun 4/60s	16 MB memory, 100 MB disk, 3.5" floppy, GX graphics coprocessor
(3) Sun 4/65s	16 MB memory, 100 MB disk, 3.5" floppy
Sun 4/75	20 MB memory, 200 & 480 MB disk, 3.5" floppy
(32) INMOS Tranputers	Parallel processing units housed in Sun 4/470

Lab software includes standard distributed SunOS/Unix software. This includes a distribution of Sun's OpenWindows, which is a version of the X windowing system. In addition, several packages have been purchased from various vendors including FrameMaker, SunGKS, SunPHIGS, SunLink DNI DECnet support software, Saber-C, DOS Windows, and Adobe Transcript. INMOS languages and development software are available for the Tranputers. The lab uses several public domain software packages including TEX, EMACS, and DECnet utilities.

The lab's software and hardware provide access to the Internet through a college-owned Merit SCP.

VAX Network

The college owns two VAX 4000 systems which serves the entire campus community for academic, administrative, and library applications. This system is accessible from eleven locations on campus which have a total of 144 stations that are publicly available for student access. In addition, there are many other terminals available in offices and laboratories across the campus.

A wide selection of software is available on the VAX systems including the Verdex VADS system for Ada software development.

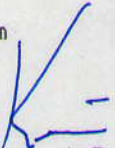
PC Network

There is a Novell local area network in the building complex where this project will be conducted that connects 49 386SX systems through a common file server. Twenty-eight of these systems are located in a computer classroom that includes 9 Epson printers, one HP Laserjet IIP printer, and a projection system.



HOPE COLLEGE

M E M O R A N D U M

DATE: June 29, 1992
TO: Herb Dershem
FROM: Kevin Kraay 
SUBJECT: Defense Advanced Research Project Award

Congratulations on your approval for a Defense Advanced Research Project grant in the amount of \$23,010 for the project entitled "Undergraduate Course Development in Software Science and ADA."

The account number 5-22655 has been assigned to this grant. Please use this number for all expenses associated with the grant.

Will you please send me a copy of the budget and Section C of the Technical Proposal for this grant. **The account number will be activated when the budget information is received and entered into the Financial Record Systems.**

The Drug-Free Workplace Act of 1988 requires Hope College to certify that we will maintain a drug-free workplace. This certification took place on the application for the NSF grant which you were awarded.

It also requires the College to provide to each employee working with a Federally sponsored program the College's policy on drugs. A copy of this policy is attached for your reference.

Please contact me if you have any questions.



HOPE COLLEGE

DEPARTMENT OF COMPUTER SCIENCE

August 13, 1992

Angela M. Coonce
Grants Officer
Defense Advanced Research Projects Agency
Contracts Management Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

Dear Ms. Coonce:

This letter is to make formal the request that I made to you in a telephone conversation last month.

I request that the DARPA grant number MDA972-92-J-1030 to Hope College have its termination date changed from June 18, 1993 to June 18, 1994. The reason for this request is that due to the late date that I was informed of the awarding of this grant, I was unable to complete the activities of the grant originally scheduled for the summer of 1992. This requires me to push the schedule of the grant back one entire year. My schedule, a revised version of that found on page 9 of my proposal, is as follows:

Date	Activity
Jan-Apr 1993	Project Director will teach course using Ada and generate ideas for laboratories.
May 1993	Design data structures course and laboratories.
June 1993	Write Ada packages, data structure visualizations, and algorithm visualizations/animations to support laboratories.
Jan-Apr 1994	Project Director will teach the newly designed course and concurrently produce the laboratory manual.
May 1994	Project Director will produce the final bound lab manual and write paper about project for SIGCSE.

If you have any questions about this request, please contact me so that we can discuss them. Thank you for all of your help on this matter. It has been a pleasure working with you.

Sincerely,

Herbert L. Dershem

MEMORANDUM

Date: August 19, 1992

To: Greg Olgers

From: Herb Dershem



Subject: Information for a news release for a grant received recently

Greg,

Here is some information about a grant that I received recently. Give me a call if you need additional information.

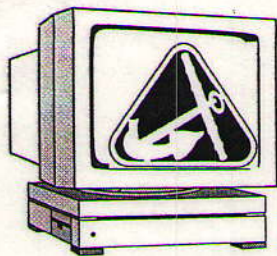
Hope College has received a grant for \$23,010 from the Defense Advanced Research Projects Agency (DARPA). The project director will be Professor Herbert L. Dershem, chairperson of the Hope College Computer Science Department. The title of the project is "Use of Ada, Laboratories, and Visualization in the Teaching of Data Structures."

The purpose of this project will be to introduce the use of the Ada language into the Hope College Data Structures course through the inclusion of a laboratory with the course. The laboratory will make extensive use of Ada and will include the illustration of data structure concepts through the use of visualization and animation. The Hope College Computer Science Sun Workstation laboratory will provide the facilities for the laboratories.

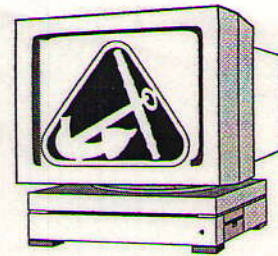
Professor Dershem will first offer the Data Structures course using Ada in the Spring 1993 semester. During the summer of 1993 he will design the laboratories to be included in the course. In the Spring semester of 1994 those laboratories will be conducted for the first time. In the summer of 1994, Professor Dershem will summarize his work in a scholarly paper and produce a laboratory manual that can be used at other institutions.

Ada was developed in the late 1970s at the initiative of the U.S. Department of Defense. The purpose was to save money by standardizing Defense software development to one language. DARPA funds the learning of Ada by students to encourage more widespread use of the language. Professor Dershem, along with Professor Michael Jipping, has written a programming languages textbook that is based on the Ada language.

This is the fifth external grant received by Hope's Computer Science Department in the past year. Four of these grants have been from federal agencies. In addition to this grant from DARPA, grants have also been received from the National Science Foundation and NASA.



Hope College
Department of Computer Science
Holland, Michigan 49422-9000
(616) 394-7510



August 22, 1995

Mr. Edward Brown
DARPA Software & Intelligent Systems Technology Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

Dear Mr. Brown:

Enclosed you will find two copies of the final report for DARPA grant MDA972-92-J-1030. The software developed under this grant has been placed in the ASSET library at West Virginia University.

I appreciate the support I received for this project. I think that you will find the results to be useful.

Please let me know if there is any other information that I can provide for you.

Sincerely,

Herbert L. Dershem, Chair

Copy: Kevin Kraay
Enclosure: 2 final reports

AdaVision and THREADS :

Algorithm Animations and Experimental Laboratories for Teaching a Data Structures Course in Ada

Cheri J. Bowsher

Dept. of Computer Science
Saint Joseph's College
Rensselaer, IN 47978

Darrick P. Brown

Dept. of Computer Science
Hope College
Holland, MI 49423

Herbert L. Dershem

Dept. of Computer Science
Hope College
Holland, MI 49423

Abstract

The overall goal of this project is to continue the implementation of a laboratory for the data structures course using Ada and algorithm visualization and animation techniques. The work done here enhances the course and contributes to the learning success of enrolled students. The first half of the project, entitled AdaVision, is an instructional aid consisting of two BTree algorithm animations. The second half of the project describes a tool called THREADS, which is used to run experiments on Ada data structures in a laboratory setting.

INTORDUCTION

The overall goal of this project is to implement a laboratory for the data structures course using Ada, algorithm visualization and animation techniques, and algorithm measurement using a tool called THREADS. Manuals have been developed to be used by students to guide their work in the laboratory. The work done in this project enhances the course and contributes to the learning success of the enrolled students.

Previous work on this project includes 6 completed algorithm animations and a basis for the THREADS program. The previously created animations include linked list, infix to postfix conversion, binary tree insert and delete, AVL tree insertions with rotations, splay tree zig-zag and zig-zig rotations, and AVL single and double rotations. Previous work on THREADS included the creation of the interface and fundamental program routines.

The philosophy used in developing the laboratory maintains that individual laboratory sessions be closed, use Ada packages, involve algorithm measurement experiments, and make use of algorithm animation. A closed laboratory means that collectively, all students have a scheduled time to work in the lab setting. An instructor is also present at this time to aid and direct their work. Many of the Ada packages are already developed, and any packages that do not already exist can be easily implemented by students. Thus, more data structures can be covered in the course. The animations help students become more familiar with algorithms and the experiments allow students to experience different qualities of the algorithms. This paper focuses on the algorithm animations and a tool used to run experiments on algorithms in the laboratory.

Laboratory Experiments: THREADS

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.

Currently, experiments are planned for the following applications:

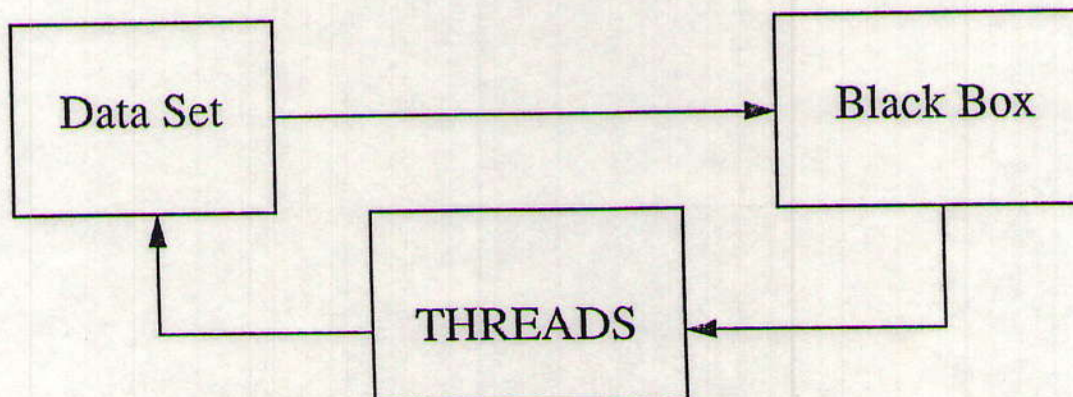
- 1) Big Oh Experiments
- 2) Comparison of different sorting algorithms
- 3) Binary search tree vs. AVL tree
- 4) Hash collision handling
- 5) Big Oh coefficient evaluation

In order to run these types of experiments in a laboratory setting, an appropriate tool is needed. The major part of our project consists of the development of such a tool, named THREADS.

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.

How the Student Uses THREADS

The basic idea behind THREADS is illustrated by the following chart



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 12. 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 generate 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.

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 supplying the write path, a notice prompt will appear and notify the user to supply THREADS with the appropriate information.

THREADS

Method: Sort 1 Write to File:

Method: Write Path:

Sample Size: 100 Use File:

Sample Distribution: Uniform Distribution Parameters

Sample Order: 0 -100 100

Table of Measurements

<u>X: Sample Size</u>	<u>Y: Measurement</u>
-----------------------	-----------------------

Figure 12

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.

- 1) Uniform.
- 2) Exponential.
- 3) Normal
- 4) Gamma
- 5) Poisson
- 6) 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 13).

Distribution Parameters

1. Exponential:	Mean: 0	
2. Normal:	Mean: 0	Standard Deviation: 5000
3. Gamma:	Order (a): 5	
4. Poisson:	Mean: 0	
5. Binomial:	Probability %: 50	n: 100

Done

(Figure 13)

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 '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.

Coefficients

If the 'Coefficients' button is clicked, a small window with five buttons will appear (Figure 14). The five buttons are $\log(n)$, n , $n\log(n)$, n^2 , and n to some power. 'n' being the sample size. If one of these buttons is clicked, an xterm 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 xterm 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 6.00000000E-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.

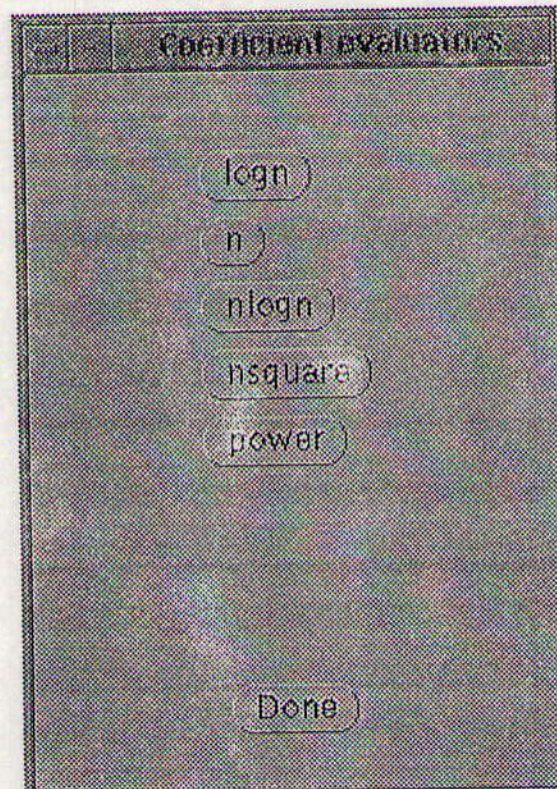


Figure 14

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.

For the Instructor

THREADS is composed of 11 files and a Makefile. The files and their contents are as follows:

threads.G:	Created by the graphical interface code generator, DevGuide 3.0.1.
threads_ui.h:	Definitions of labels used to receive information about the UIT objects.
threads_ui.cc	Sets up the interface, and begins waiting for events to process.
threads_stubs.cc:	Callback functions for the various widgets on the interface. Also contains any auxiliary functions needed by THREADS.
threads.info:	Contains the help text retrieved by pressing the "Help" key on the keyboard.
longhelp.info:	Contains the complete help text displayed in the THREADS help window.
threads.icon:	Contains the graphical data for the THREADS icon.
threads.mask	Contains the graphical data for the THREADS icon mask.
xvgr.prefs:	Contains the preferences for xvgr.
coef.scpt:	Contains the script that is called when one of the coefficient buttons are clicked.
arret.cc:	A very small program that just waits until the user presses the Return key; used in coef.scpt.

Black boxes may be added to THREADS with little effort. The spots where the code needs to be modified are marked by comments of the form:

```
/* NEW METHODS: add any new methods here */
```

First, in the file threads_ui.cc, insert the line:

```
(void) mthdchoice.addChoice ("Method name");
```

where 'Method name' is the name to appear on the menu in the interface. For each new method added, a similar line must be inserted. The lines need to be added to the current listing of choices, which is marked in the code.

Next, in the file threads_stubs.cc, add another "else if" condition of the form:

```
else if (strcmp(method, "MethodFile") == 0) {  
    strcat(command, "MethodName");  
}
```

In this section 'MethodFile' is the executable file name for the black box. Each new method needs to have its own case in the "if-else if" statement.

New methods can also be used by placing them in the blackbox directory and typing the name of

the method executable in the "Method:" text field. This is much easier from a programming aspect, but this requires the user to type in the name instead of being able to select it from the menu.

New types of distributions may be added in a similar fashion. The menu choices for "Sample Distribution" are added the same way as those for "Method." A clear definition of what the distribution means and how it will affect random generation of integers will facilitate the changes in the code of `threads_stubs.cc`.

In `threads_stubs.cc`, the system calls use the full path names to execute the black box and to use the other include files. Check that these paths are correct, and change them as appropriate.

CONCLUSION

As future work on this project, further AdaVision animations could be created or animations which exist as part of the XTANGO package could be refined for the purpose of implementation into a data structures course. In regard to THREADS, further use of the "Sample Distribution" feature is yet to be found, and additional features may be added.

The use of AdaVision and THREADS in a laboratory setting is intended to involve students more directly in their instruction than a classroom setting alone. The animations are intended to improve students' understanding of the data structures and algorithms being taught, and performing experiments on the algorithms allow the students to analyze and experience the effectiveness and efficiency of different solutions to problems. It is hoped that this project does indeed enhance a data structures course, helping instructors convey to students the abstract concepts of the course and actively involve them in the instruction.

BIBLIOGRAPHY

Donovan, Tommy. "Getting Started with TANGO." 1990.

Press, William H., Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, 1986.

Stasko, John T. "BWE Programmer's Manual: TANGO." Brown University, 1990.

Stasko, John T. "TANGO: A framework and system for algorithm animation." *Computer* Vol. 23, No. 5, p. 27-38, 1990.

Turner, Paul J. "ACE/gr User's Manual: Graphics for exploratory data analysis." Center for Coastal and Land-Margin Research, Oregon Graduate Institute of Sci. and Tech., 1992.

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

Algorithm Animations: AdaVision

AdaVision serves as a teaching tool for data structures courses taught in Ada by allowing students to visualize data structure concepts, rather than try to learn and understand the concepts in an abstract manner. Using the algorithm animation package XTANGO, two-dimensional, color animations are created by combining two files. The first file contains the implementation of the algorithm being animated along with procedure calls to the second file, which consists of the animation scenes. The structures and data involved in the algorithm implementation are represented in the animation scenes by images which move as a result of interesting events, such as the insertion of a node into a linked list.

Some of the previous animations created for the AdaVision project display the corresponding Ada code on the XTANGO window as the algorithm is being animated. This display of code enables students to view the connection between Ada code and the action of algorithms on data and data structures. At the beginning of an animation, the first line of Ada code is highlighted by a rectangular image. Succeeding lines of code are then highlighted as they are executed within the algorithm implementation, allowing students to observe the movement of images taking place in conjunction with the highlighted code.

The two most current AdaVision animations illustrate insertions and deletions on a BTree. Due to the complexity of the BTree algorithm, these animations do not include the corresponding Ada code as part of their displays. Also due to complexity, and because the implementation is not necessary for what the animations are intended to illustrate, the actual BTree algorithm is not implemented into the file which normally would contain the algorithm implementation. Instead, each animation merely demonstrates the possible actions that would be performed on a particular BTree if an insertion or deletion were to occur.

Within each of the BTree animations, as a value is inserted or deleted, images which represent nodes, links and values are repositioned accordingly. Nodes containing values that are not relevant to the particular BTree demonstration are represented by empty, smaller-sized, rectangular images, while nodes containing values which are necessary for the demonstration are represented by non-empty, larger-sized, rectangular images. The condition of the particular BTree that is shown depends on the user's response to a series of questions regarding the BTree he or she would like to view. These questions are asked in the form of a dialogue before an animation is shown. Once the animation is displayed, a label describing the insertion or deletion being performed is written near the bottom of the XTANGO window. Also, in the upper left corner of the window is a display of text which indicates the value being inserted or deleted. The BTree deletion animation additionally includes a short text phrase which explains what is occurring within the BTree at the moment that the images are moving.

BTree Insertion

The BTree insertion animation illustrates four possible ways for a value to be inserted into a BTree: into a non-full leaf, a leaf whose parent is non-full, whose ancestor is non-full, or into the leaf of a BTree which does not fit any of these cases, in which case the root is split. The anima-

tion contains a dialogue which asks the user questions regarding the type of BTree into which he or she would like to see an insertion performed. Depending on the user's response to the questions, the appropriate insertion simulation is shown.

The case in which the value "100" is inserted into a leaf whose parent is non-full is illustrated below.

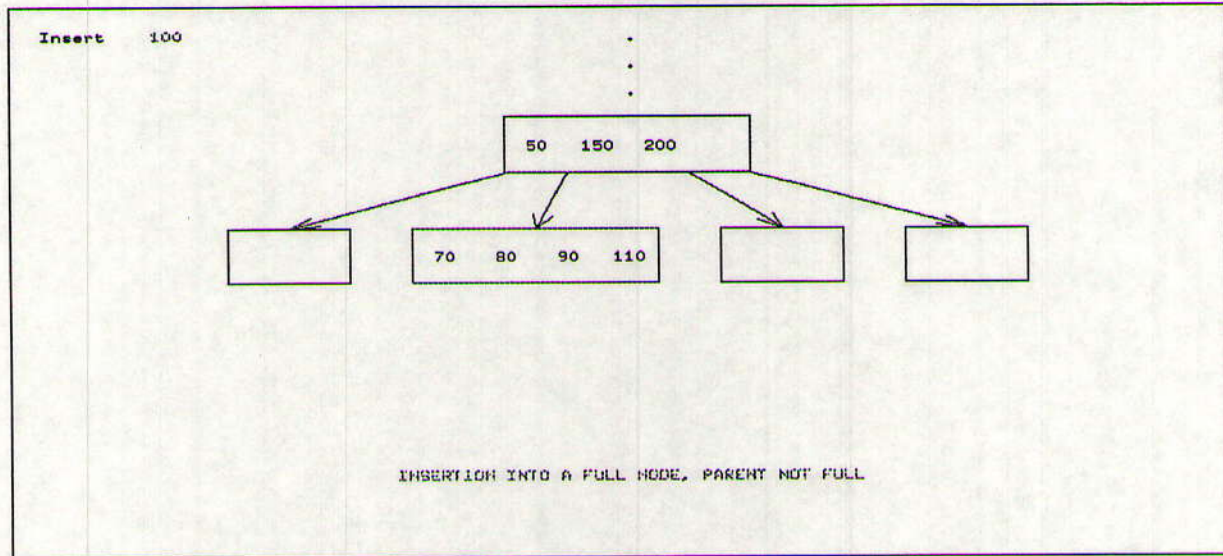


Figure 1. The original form of the BTree is displayed.

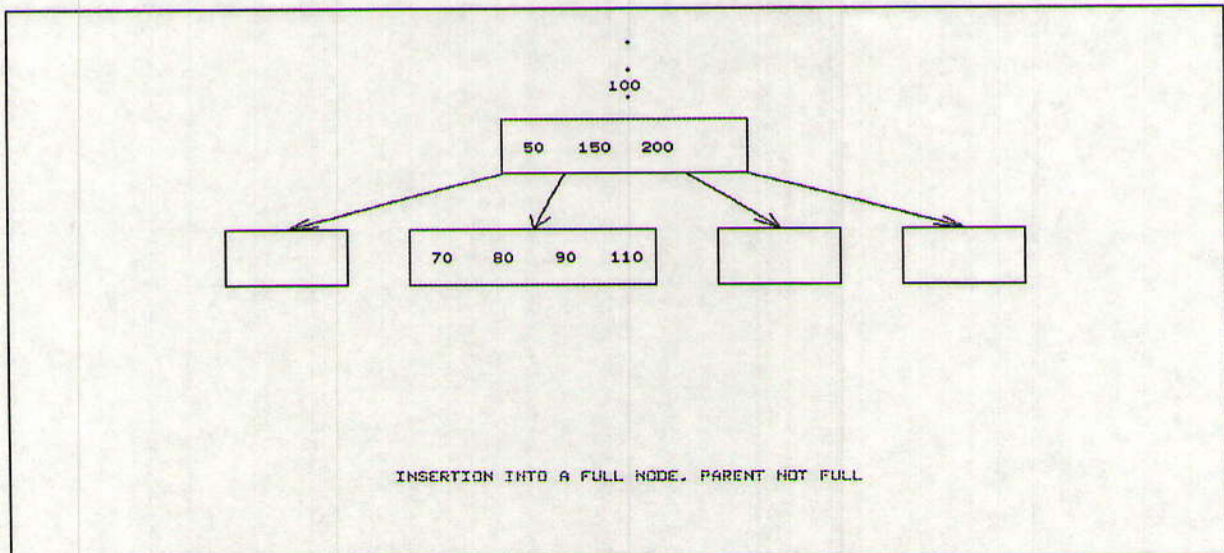


Figure 2. The value "100" moves down through the top of the tree.

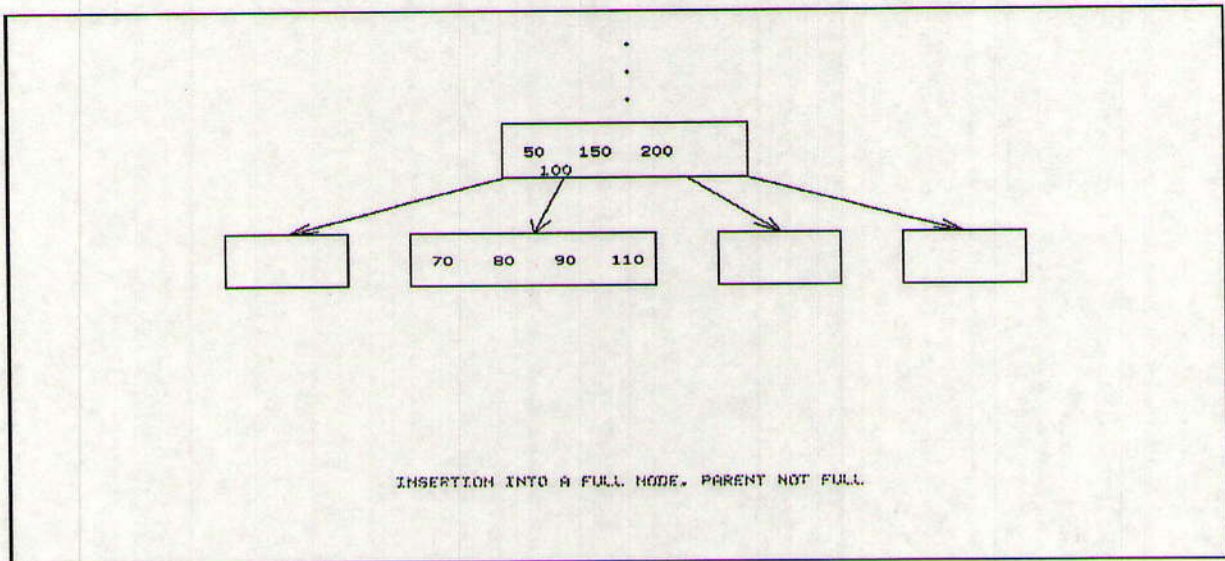


Figure 3. The value "100" moves down through the parent node.

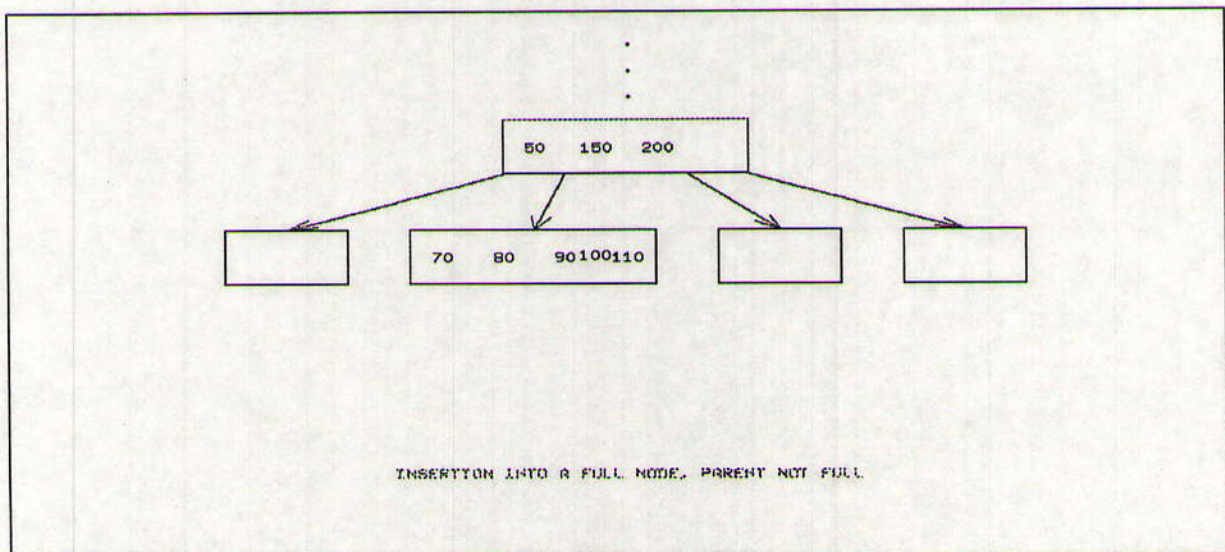


Figure 4. The value "100" moves into its correct place in the leaf.

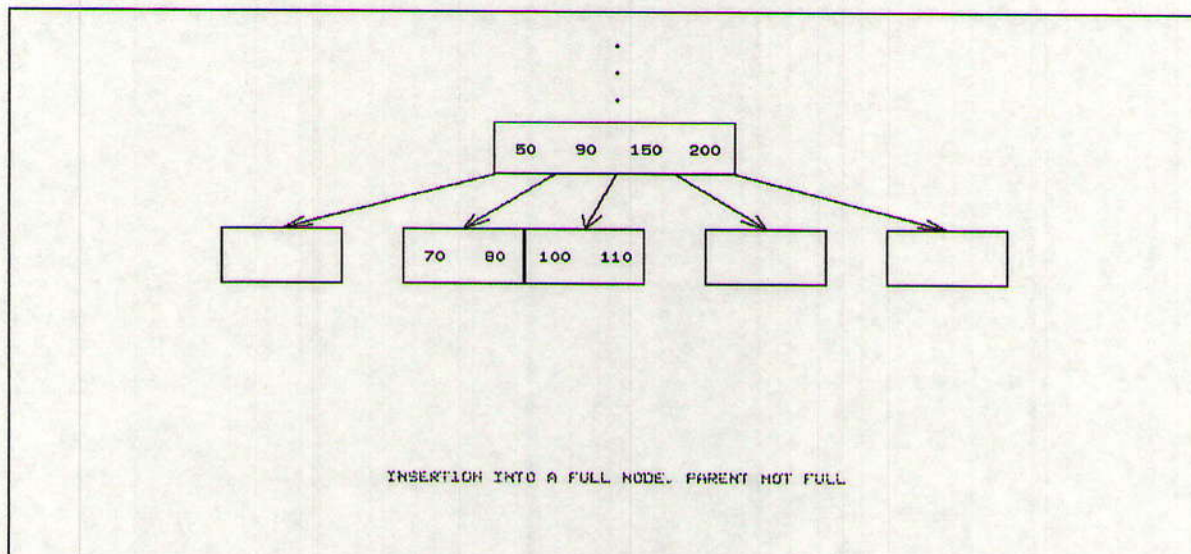


Figure 5. The middle value of the leaf, "90", moves up to the parent node and the leaf is split.

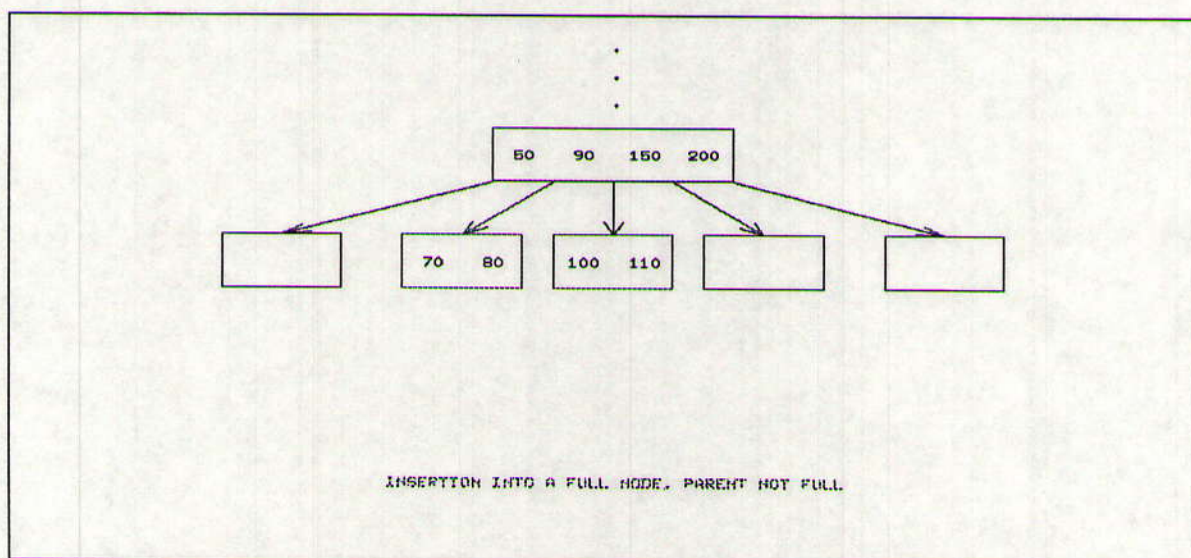


Figure 6. The split nodes, along with their values and links, are repositioned.

BTree Deletion

The BTree deletion animation illustrates six possible ways for a value to be deleted from a BTree: from a non-leaf, a leaf that is larger than minimum size, a leaf whose neighbor is larger than minimum size, whose parent is larger than minimum size, whose ancestor is larger than minimum size, or a deletion from a BTree which does not fit any of these cases, in which case the root is dissolved. Similar to the BTree insertion, this animation contains a dialogue of questions. Depending on the user's response to the questions asked, the appropriate deletion simulation is shown. The deletion of a value from the tree is illustrated as the value moves down and into a "garbage can" image.

The case in which the value "100" is deleted from a leaf whose parent is larger than the minimum size is illustrated below.

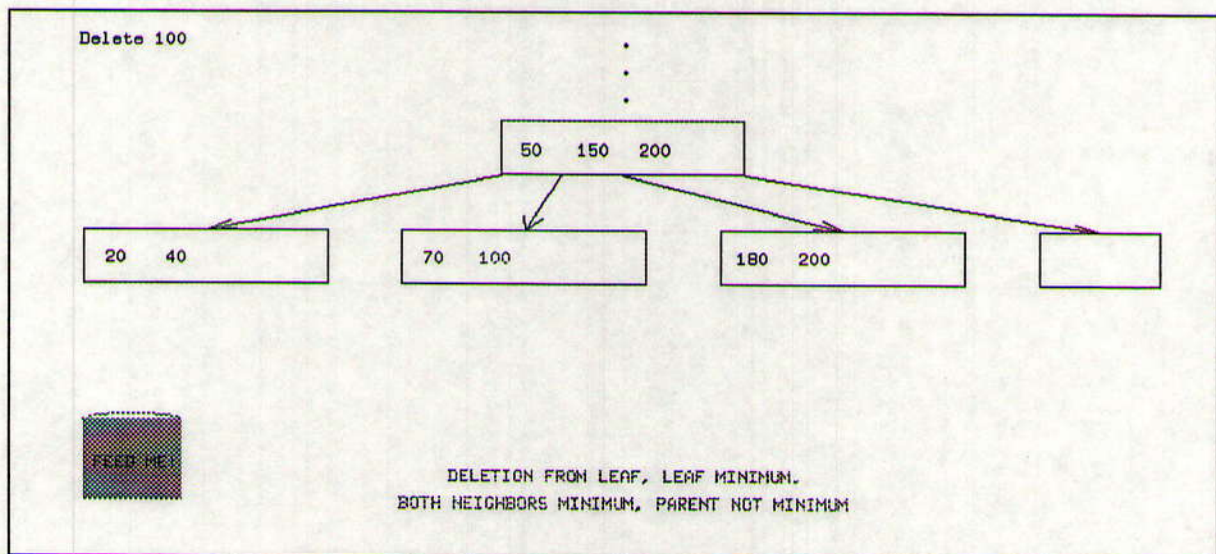


Figure 7. The original form of the BTree is displayed.

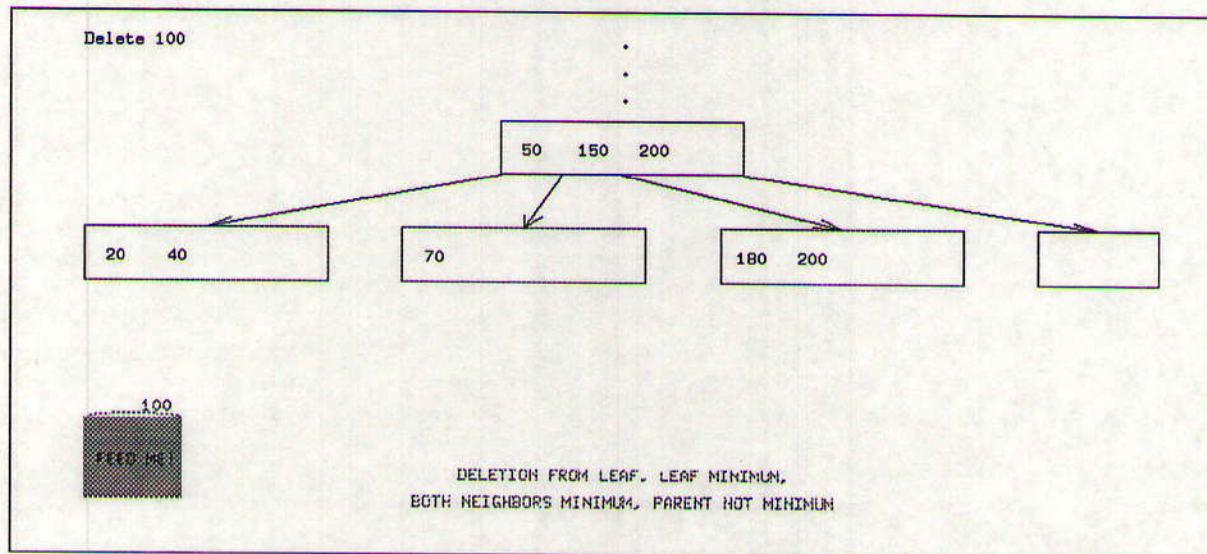


Figure 8. The value "100" is moved into the deletion "garbage can."

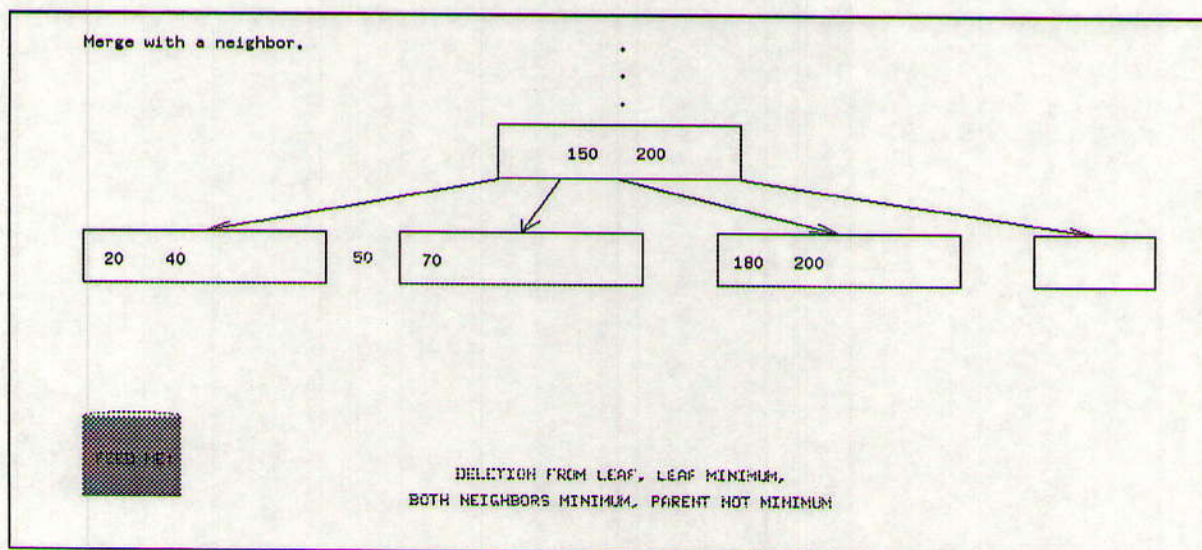


Figure 9. The value "50" is moved down so that the leaf may regain its order.

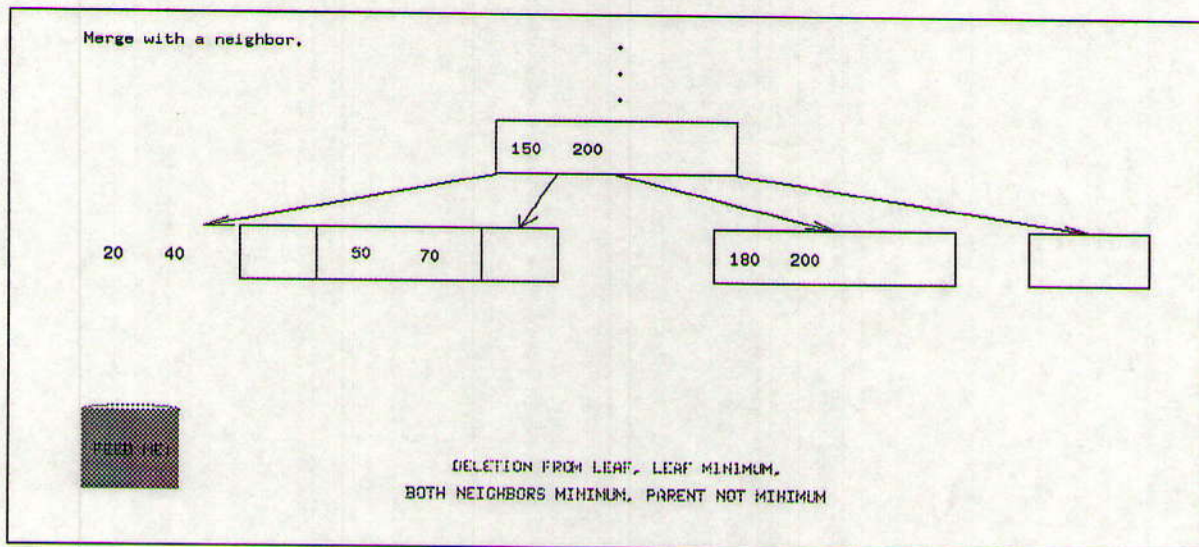


Figure 10. The leaf and its neighbor merge.

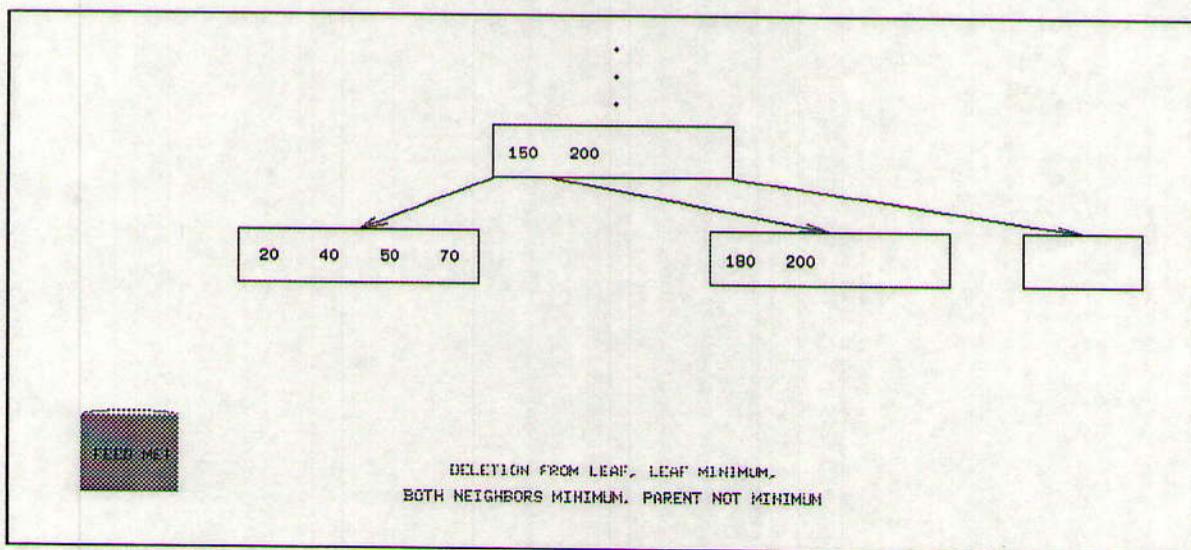


Figure 11. The two nodes, their links, and values as repositioned as one.

Data Structures with Ada Packages, Laboratories, and Animations

Herbert L. Dershem

Wendy L. Barth

Cheri J. Bowsher

Darrick P. Brown

1.0 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.

2.0 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 are 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].

3.0 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 use in the laboratory. The animations are intended to enhance student understanding of algorithms, particularly since ~~they~~ the students do not write code to implement the algorithms in most cases. Most 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

A more detailed description of these animations along with illustrations are found in Appendix A.

4.0 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 Appendix B.

Descriptions of the laboratory exercises are found in Appendix C. The titles and brief descriptions are given below.

1. Writing an Ada program

The students are introduced to the Ada language by writing a program to compute the n th 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.

5.0 Project Activities

The project was conducted on the following time table:

1. June-July 1993
Course and laboratories designed, Ada packages written, data structure visualizations constructed to support laboratories. Professor Dershem was assisted by Wendy Barth and Cheri Bowsher with support from a National Science Foundation Research Experiences for Undergraduates program grant. Also, student Bob Chen assisted.
2. November 1993
Results of work of previous summer were presented by Wendy Barth and Cheri Bowsher at the Argonne National Laboratories Symposium on Undergraduate Research.
3. March 1994
Seminar on algorithm animation was presented at the United States Air Force Academy discussing animations developed during the previous summer.
4. June-July 1994
Course materials finalized. Documentation prepared for software products. Professor Dershem was assisted by Cheri Bowsher and Darrick Brown with support from a National Science Foundation Research Experiences for Undergraduates program grant.
5. Fall Semester 1994
Professor Dershem was provided 1/3 release time by Hope College for the preparation of the course materials in the teaching of the course.
6. November 1993
Results of work of previous summer were presented by Darrick Brown and Cheri Bowsher at the Argonne National Laboratories Symposium on Undergraduate Research.
7. Spring Semester 1995
Professor Dershem used the final materials in CSCI 286, Data Structures, at Hope College.
8. August 1995
Final report was prepared.

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.

Appendix B

AdaVision - Visualization & Animation

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. With the exception of the AVL insertion, 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.

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 will occur 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.

```
Prev_Cell := Find(input_value, L);
Temp := new Node"(X, Prev_Cell.Next);
Prev_Cell.Next := Temp;
```

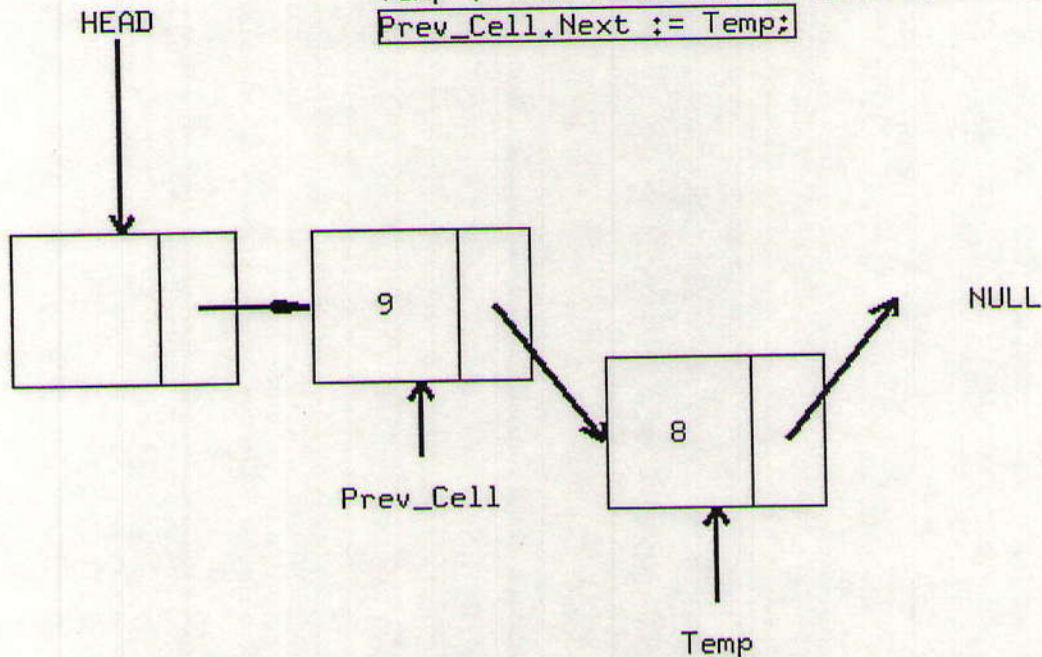


Figure 1. Inserting a node at the start of a linked list

Infix to Postfix Conversion

AdaVision's 'postfix' animates an infix to postfix conversion. After clicking XTANGO's 'Run Animation' button, the user is asked to enter an infix expression. This original stream of characters appears below the label 'INPUT' in the XTANGO window. The postfix expression is built and appears under the label 'OUTPUT' and the operators are stored in a 'STACK' image as they are processed. Throughout the animation, an arrow is used to point to the character of the input stream which is currently under consideration. The corresponding lines of Ada code are also displayed and highlighted as the conversion is performed. Execution is completed when all symbols in the infix expression have been processed and the stack is empty.

It is assumed in this animation that the user has a previous understanding of stack operations, such as pop and push. Once a character, or symbol, is read, it is pushed onto either STACK or OUTPUT. An operand in the infix expression moves directly from the input to the output stream. If an operator is encountered, it is moved to the output after all operators with a lesser precedence are popped from the stack and pushed onto the output. A comparison between the operator being considered in the input and the operator at the top of the stack is indicated by a blinking top-of-stack element.

CONVERSION FROM INFIX TO POSTFIX

OUTPUT

CONVERSION FROM INFIX TO POSTFIX

OUTPUT

page 8

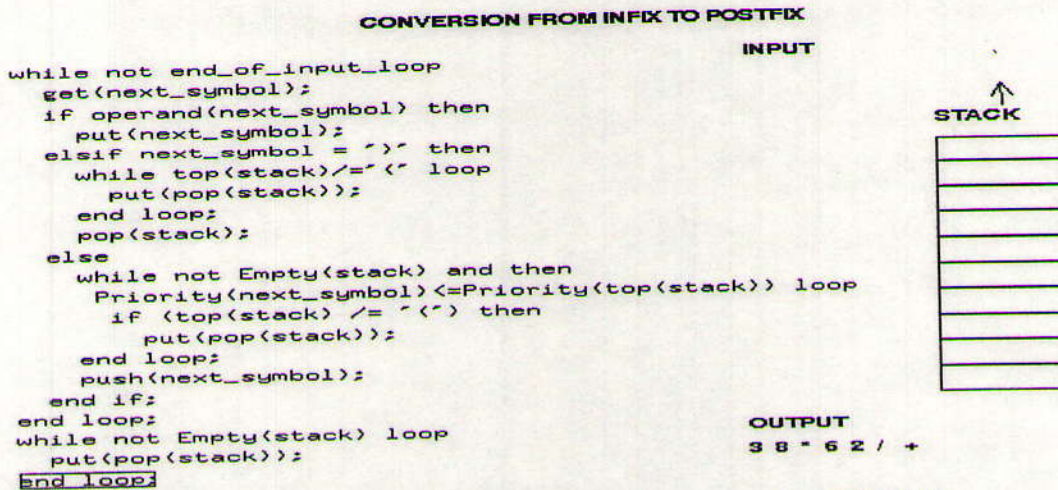


Figure 4: The final postfix expression appears in OUTPUT.

Binary Search Tree

AdaVision's "bintree" animation demonstrates insertions and deletions on a binary search tree. The tree is displayed on the right side of the display area, and the Ada code is shown on the left side.

Upon beginning the animation, the user is asked whether or not s/he wants to see comparisons. If the user answers affirmatively, all comparisons will be animated during the viewing of the animation. The are animated immediately before a value moves down the tree as follows: if the insertion value is less than the existing node, it moves to the left of the node and a less-than sign appears between the values. If the new value is greater than the existing node, it moves to the right. If the user does not wish to see the comparisons, s/he should answer no. After making this decision, a menu appears in the shell window. The user inputs which operation s/he would like to see, along with the value to be inserted into or deleted from the tree.

The Ada code for both insertions and deletions is recursive. Each level of recursion is shown by outlining the tree currently under consideration. Old outlines 'dim' by changing color when a new level of recursion is entered. All code is erased and rewritten to show each recursive call.

When an insertion is animated, the new value appears in the upper-left corner and then moves to the root position. The new value slides down to the appropriate child position: left if less, right if greater than the node. This continues until the new value finds an empty position or encounters a node of the same value. If the new value finds an empty position, a new link is drawn to connect it to the tree. No value may appear more than once in the tree, so if the user does try to insert a value that already exists in the tree, the node will flash, and the new copy of the value will be moved out of the tree.

A deletion is animated in similar fashion, except that we are searching to match the deletion value. Once the value has been found, the correct replacement flashes, the node being deleted is 'lifted' out of the tree, and the replacement node or sub-Tree is 'pulled' into place by the link that

previously pointed to the deleted node. If no node is found that matches the deletion value, the deletion value will be 'lifted' out of the tree.

```
Insert(X, T)
```

```
if T = null then
```

```
  T := new Tree_Node(X, null, null);
```

```
elseif X < T.Element then
```

```
  Insert(X, T.Left);
```

```
elseif X > T.Element then
```

```
  Insert(X, T.Right);
```

```
end if;
```

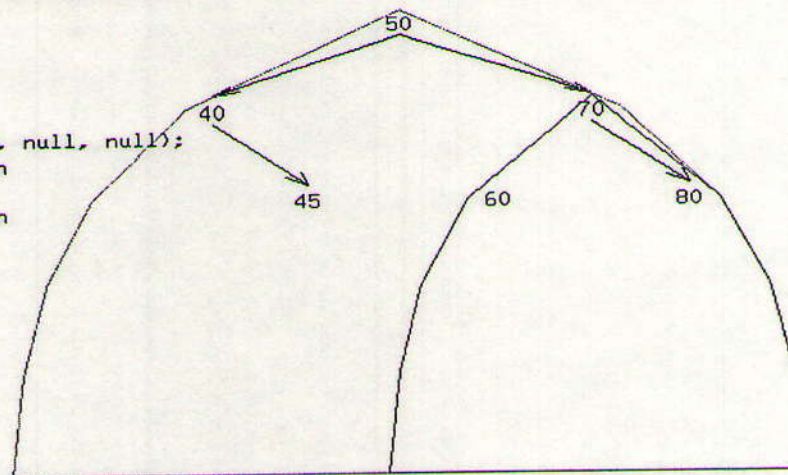


Figure 5. Inserting a node into a suB-Tree

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 suB-Trees 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 suB-Trees of depth n, and two triangular suB-Trees of depth 'n+1'. The nodes in each rotation are connected to suB-Trees 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 suB-Tree, potentially causing the tree to become unbalanced. The element is inserted into the left-most suB-Tree of the pivot for the single left rotation, the right-most suB-Tree for the single right rotation, the right suB-Tree of the left child of the pivot for the double left rotation, and the left suB-Tree 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.

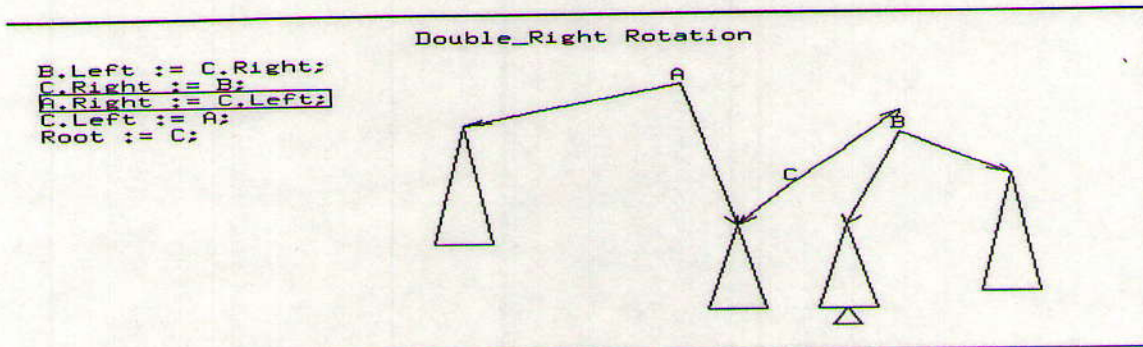


Figure 6 . The repositioning of links due to the insertion of an element into a suB-Tree.

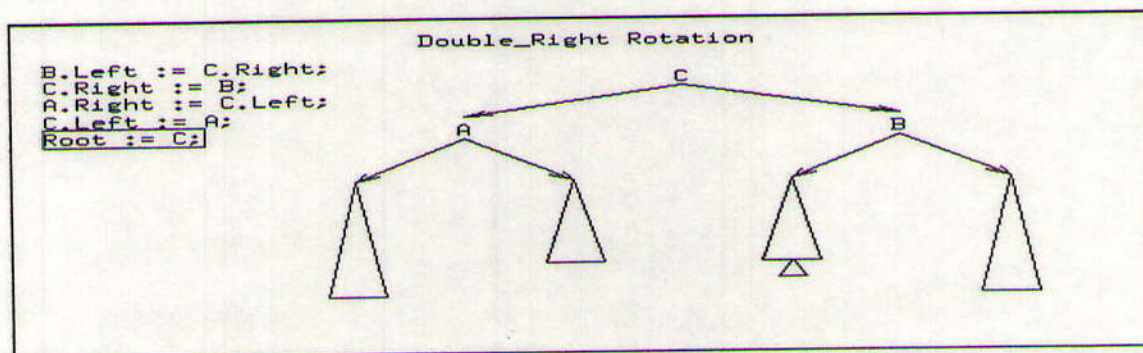


Figure 7. A balanced AVL tree following the performance of a double right rotation.

AVL Tree Insertion

AdaVision's "avlins" animation demonstrates insertions into an AVL tree. This animation assumes that the user already has a working knowledge of the four types of AVL rotations: single left, single right, double left, and double right. No code is displayed in the animation window, but the user is notified which rotation is being performed.

After pressing the 'run animation' button on the XTANGO window, the user is prompted for the value to be inserted into the tree. Values may be between 1 and 1000, with 0 serving as the quit option. The user inputs the value s/he would like to see inserted, and the animation proceeds from there.

First, the insertion into the tree is animated. The new value appears in the upper left corner of the display window, while the root of the tree is surrounded by a blue circle. This blue lozenge marks the position in the tree currently being compared with the new value, and will eventually mark the spot where the value is to be placed. Next, the circled node flashes to show that it is being compared to the insertion value. If the new value is less than or equal to the surrounded node, the lozenge moves to the position of the left child of the node. If the new value is greater than the node, the lozenge moves to the right child's position. After the correct move of the lozenge, a new comparison is done. If no node exists to compare the value with, the value is inserted into the tree as a new node, and an edge is added to connect it to the tree.

If the new node causes the tree to be unbalanced, a rotation is animated. First, we search for the unbalanced suB-Tree. The blue lozenge is replaced by a green one, and the green lozenge moves up the tree until it finds a node whose tree is unbalanced. If one is found, the user is notified of which of the four rotations will occur, and the rotation is done. If one is not found, the green lozenge disappears.

In a rotation, when a node moves, the edge pointing to it is deleted before the movement occurs. After the node has assumed its new position, a new edge is drawn that points to it. The only exception to this action is when the node is moving from or to the root position in the tree.

After the value has been inserted and any necessary rotation done, the user is prompted for the next value to insert into the tree.

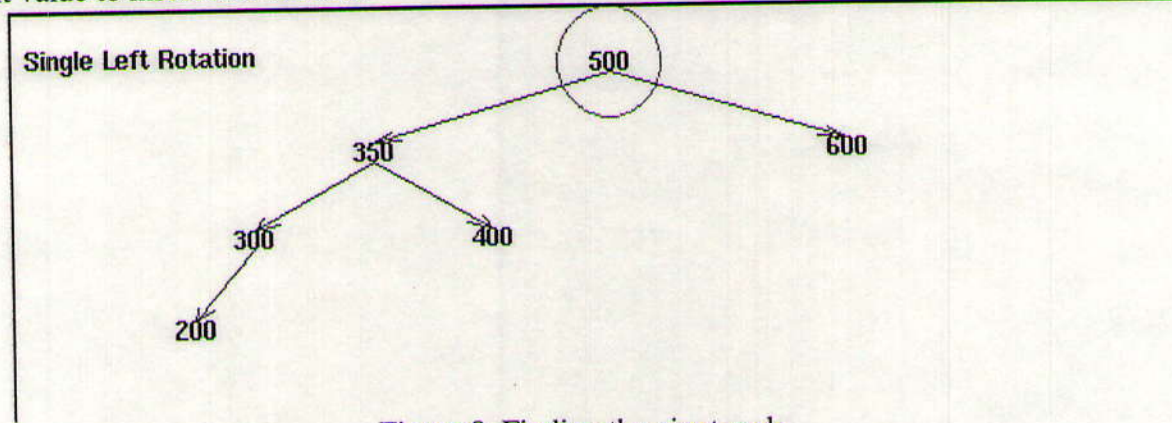


Figure 8. Finding the pivot node

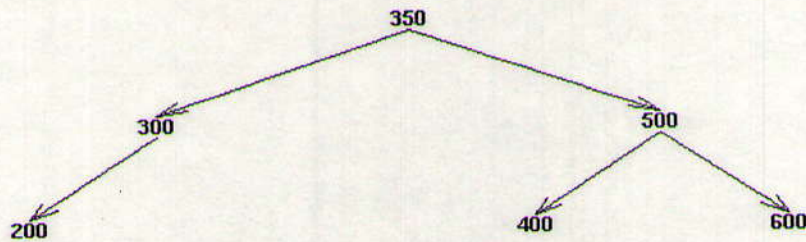


Figure 9. The balanced tree following a single left rotation

Splay Tree Rotation

AdaVision's 'splay' demonstrates zig-zig and zig-zag splay tree rotations. The user may view either of the two rotations as many times as s/he would like by clicking XTANGO's 'Run Animation' button and choosing an option number from a shell window.

The zig-zag demonstration includes three nodes: X, the left child of the pivot's right child and the node to be accessed; G, the original root and grandparent node of X; and P, the parent node of X. These nodes are connected by 'links' to four triangular suB-Trees labelled A,B,C, and D, where A is the left-most suB-Tree and D the right-most suB-Tree.

In viewing the zig-zag animation, the links are repositioned one at a time and the corresponding lines of Ada code are highlighted until the objective of making X the new root of the tree is accomplished. Following the series of link movements, the images in the tree are reconstructed, bringing the tree into its rotated form.

The zig-zig rotation is demonstrated similarly, except that X is initially positioned as the left child of the pivot's left child.

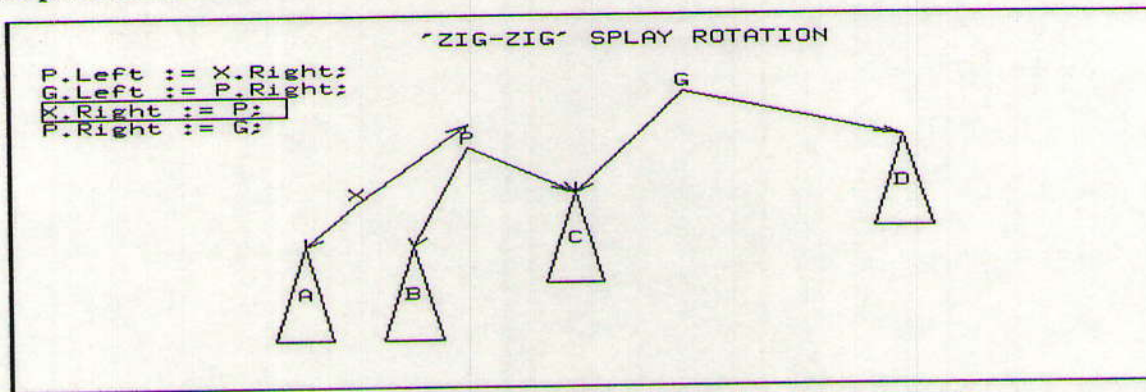


Figure 10. The movement of links in order to access node X.

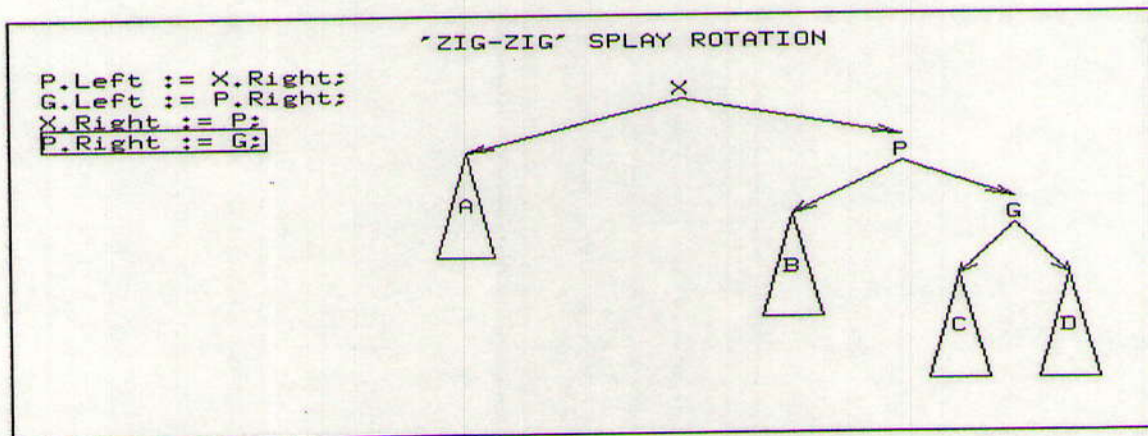


Figure 11. The rotated splay tree with X as its new root.

B-Tree Algorithms

The following two AdaVision animations illustrate insertions and deletions on a B-Tree. Due to the complexity of the B-Tree algorithm, these animations do not include the corresponding Ada code as part of their displays. Also due to complexity, and because the implementation is not necessary for what the animations are intended to illustrate, the actual B-Tree algorithm is not implemented into one of the two separate XTANGO files as it normally would be. Instead, each animation merely demonstrates the possible actions that would be performed on a particular B-Tree if an insertion or deletion were to occur.

Within each of the B-Tree animations, as a value is inserted or deleted, images which represent

nodes, links and values are repositioned accordingly. Nodes containing values that are not relevant to the particular B-Tree demonstration are represented by empty, smaller-sized, rectangular images, while nodes containing values which are necessary for the demonstration are represented by non-empty, larger-sized, rectangular images. The condition of the particular B-Tree that is shown depends on the user's response to a series of questions regarding the B-Tree he or she would like to view. These questions are asked in the form of a dialogue before an animation is shown. Once the animation is displayed, a label describing the insertion or deletion being performed is written near the bottom of the XTANGO window. Also, in the upper left corner of the window is a display of text which indicates the value being inserted or deleted. The B-Tree deletion animation additionally includes a short text phrase which explains what is occurring within the B-Tree at the moment that the images are moving.

B-Tree Insertion

The B-Tree insertion animation illustrates four possible ways for a value to be inserted into a B-Tree: into a non-full leaf, a leaf whose parent is non-full, whose ancestor is non-full, or into the leaf of a B-Tree which does not fit any of these cases, in which case the root is split. The animation contains a dialogue which asks the user questions regarding the type of B-Tree into which he or she would like to see an insertion performed. Depending on the user's response to the questions, the appropriate insertion simulation is shown.

The case in which the value "100" is inserted into a leaf whose parent is non-full is illustrated below.

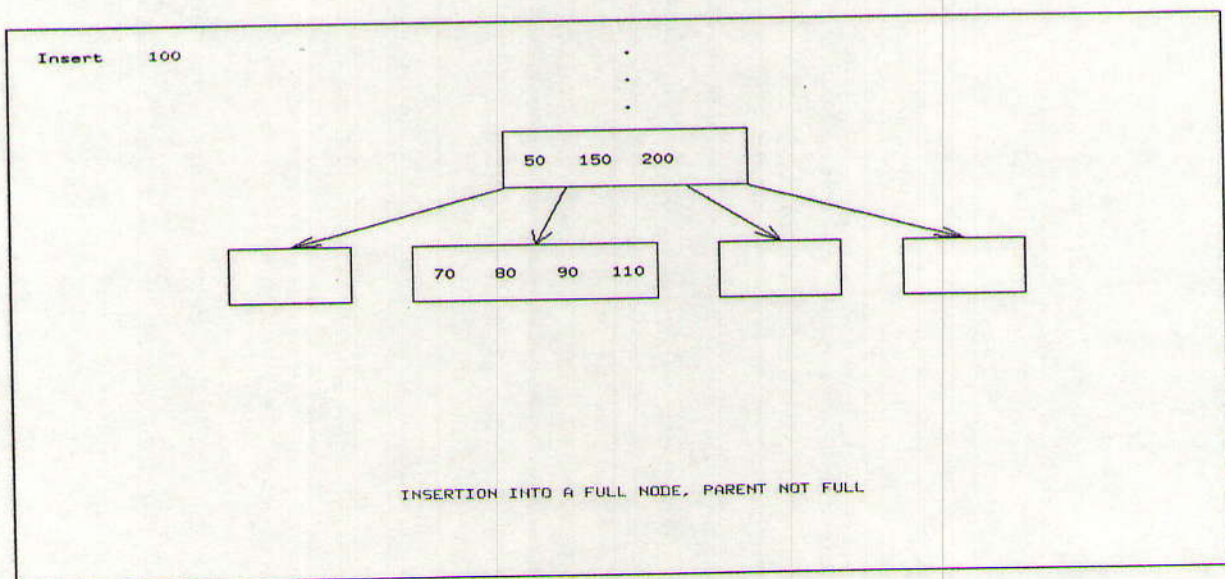


Figure 12. The original form of the B-Tree is displayed.

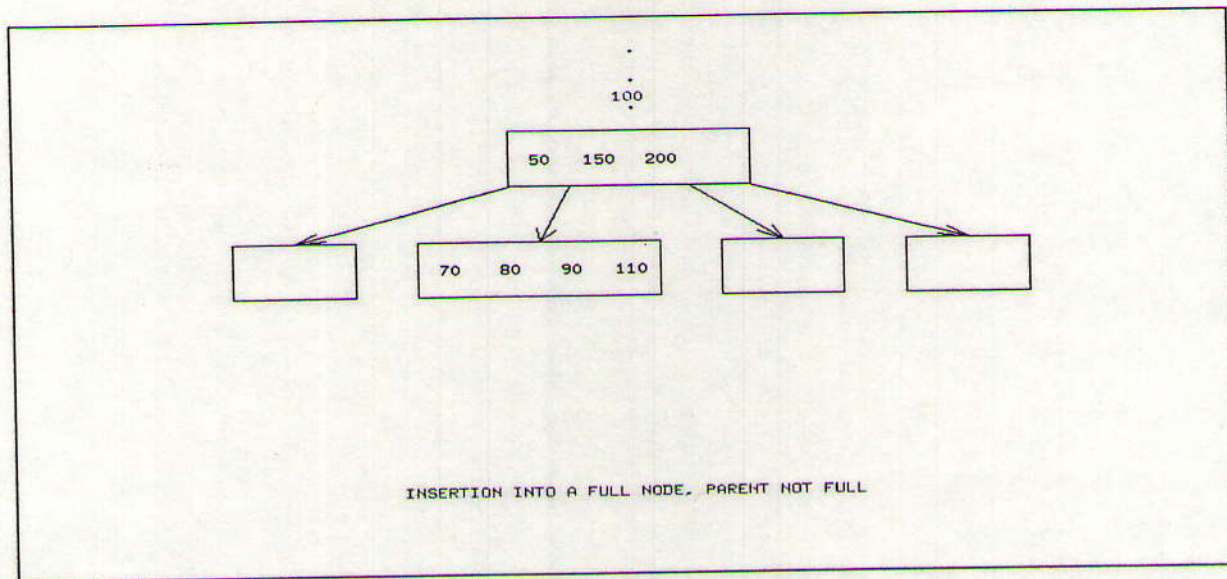


Figure 13. The value "100" moves down through the top of the tree.

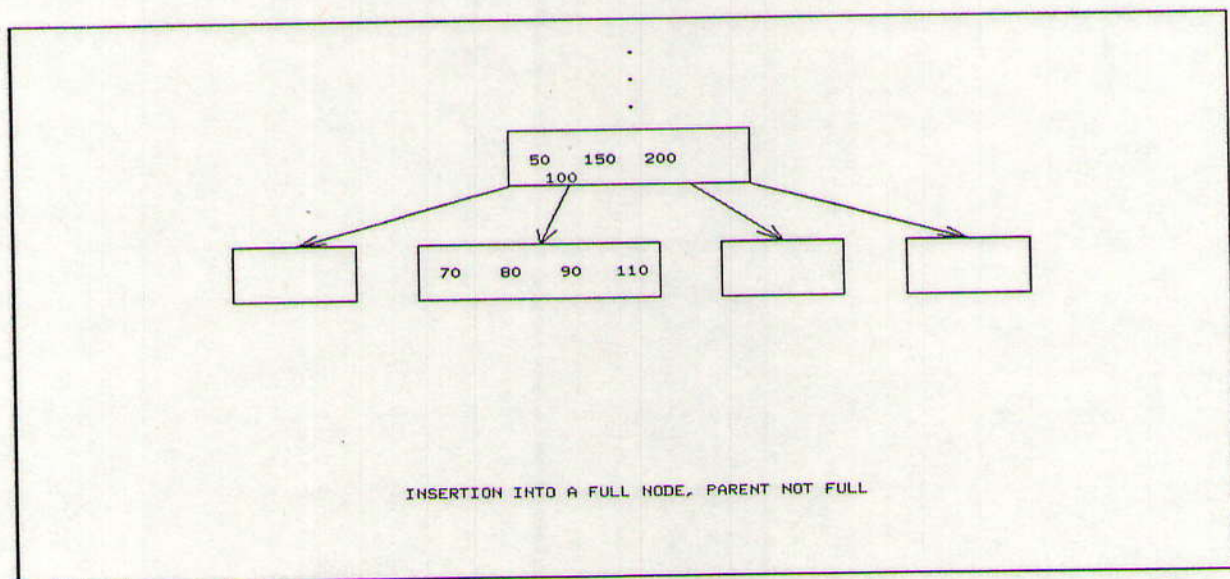


Figure 14. The value "100" moves down through the parent node.

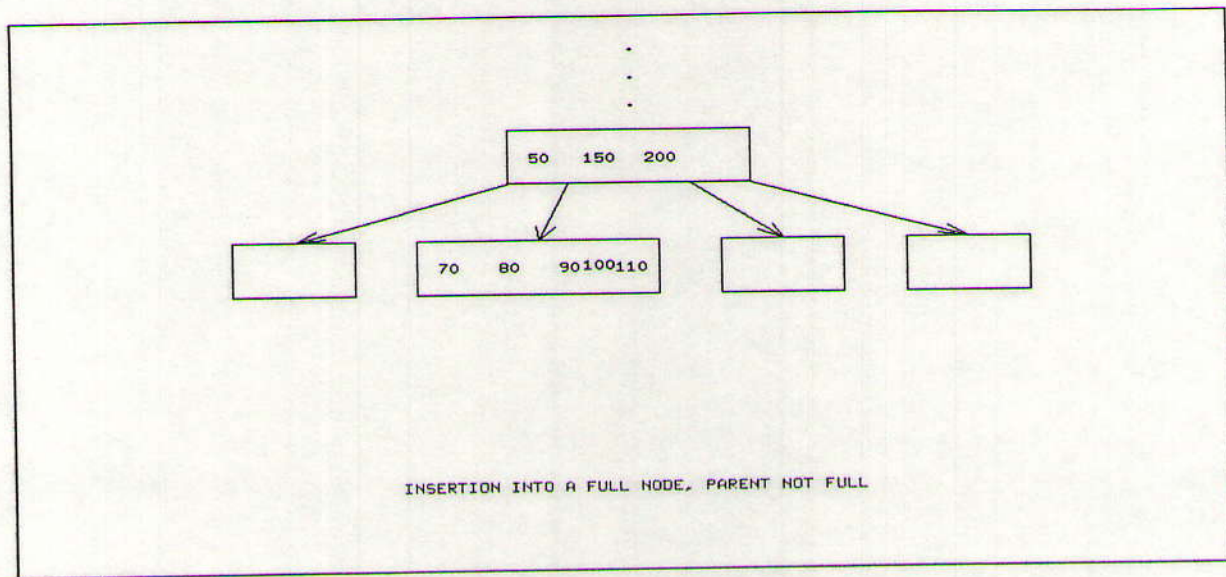


Figure 15. The value "100" is inserted into its correct place in the leaf.

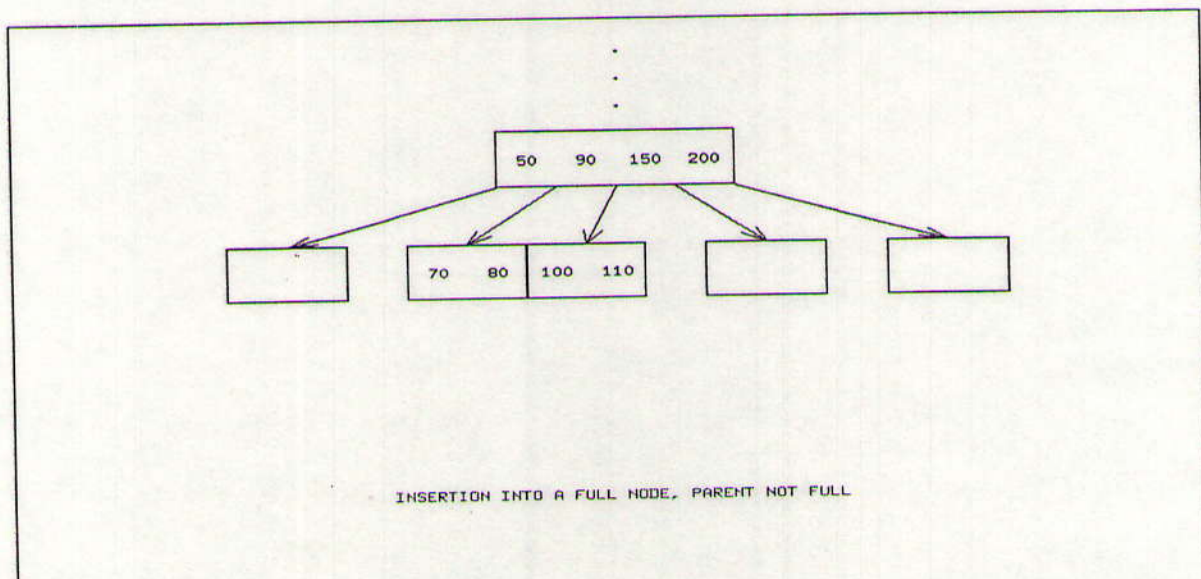


Figure 16. The middle value of the leaf, "90", moves up to the parent node and the leaf is split.

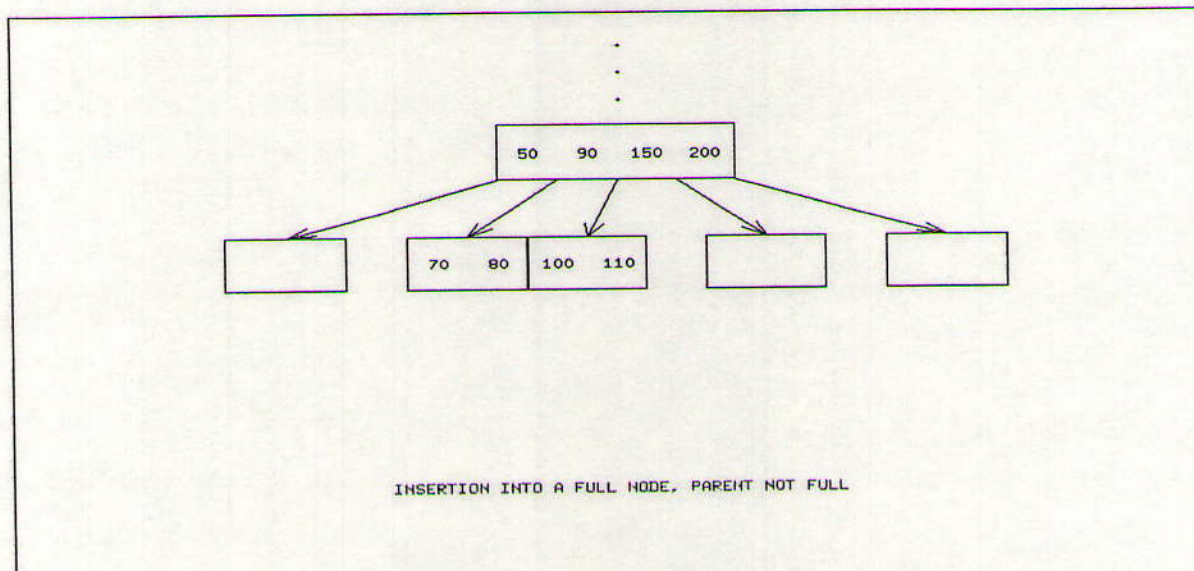


Figure 17. The split nodes, along with their values and links, are repositioned.

B-Tree Deletion

The B-Tree deletion animation illustrates six possible ways for a value to be deleted from a B-Tree: from a non-leaf, a leaf that is larger than minimum size, a leaf whose neighbor is larger than minimum size, whose parent is larger than minimum size, whose ancestor is larger than minimum size, or a deletion from a B-Tree which does not fit any of these cases, in which case the root is dissolved. Similar to the B-Tree insertion, this animation contains a dialogue of questions. Depending on the user's response to the questions asked, the appropriate deletion simulation is shown. The deletion of a value from the tree is illustrated as the value moves down and into a "garbage can" image. The movement of remaining values, nodes and links then proceeds as their images are repositioned accordingly.

The case in which the value "100" is deleted from a leaf whose parent is larger than the minimum size is illustrated below.

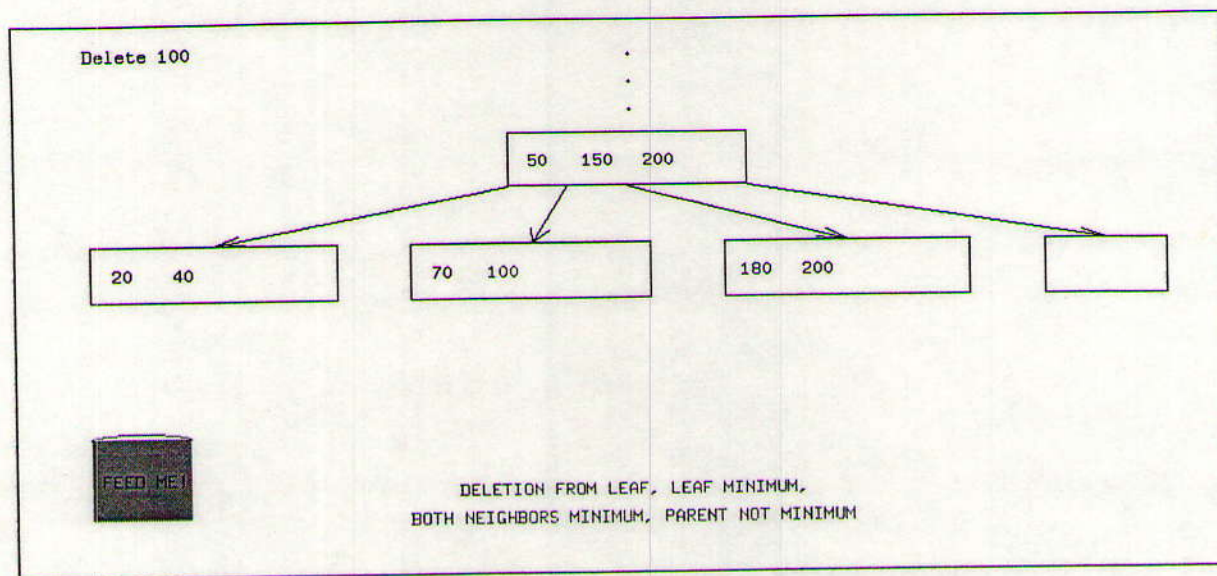


Figure 18. The original form of the B-Tree is displayed.

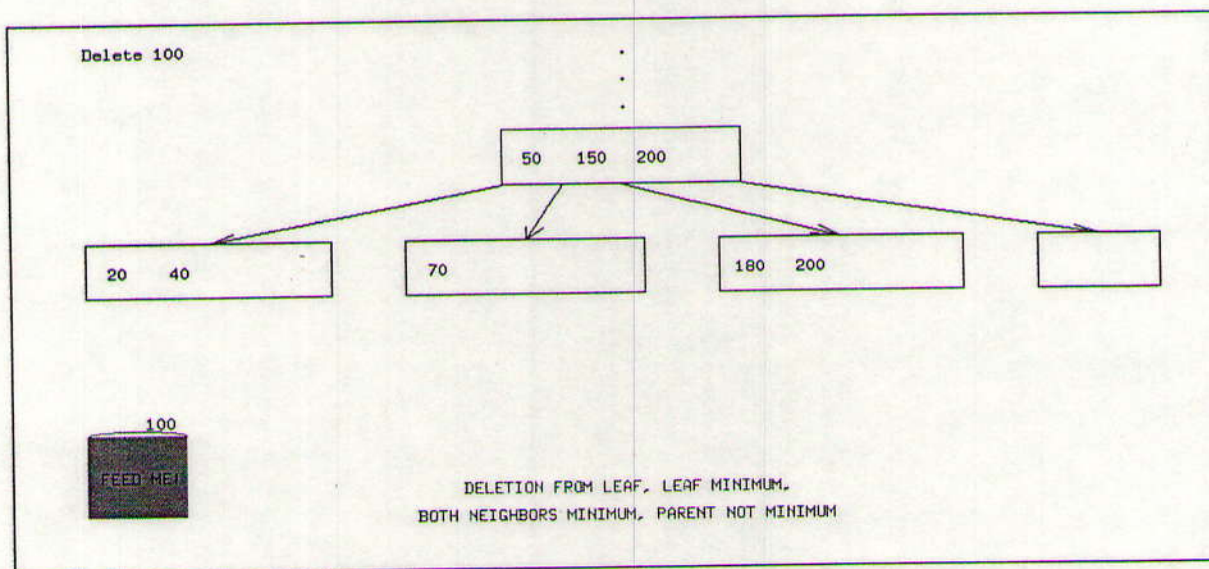


Figure 19. The value "100" is deleted by being thrown into the "garbage can."

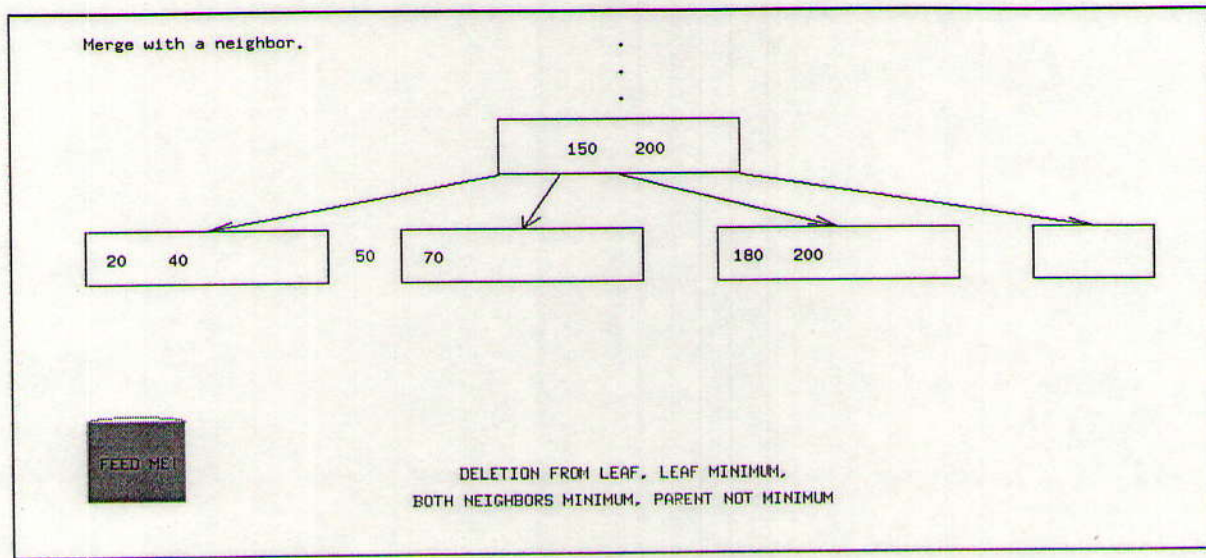


Figure 20. The value "50" is moved down so that the leaf may regain its order.

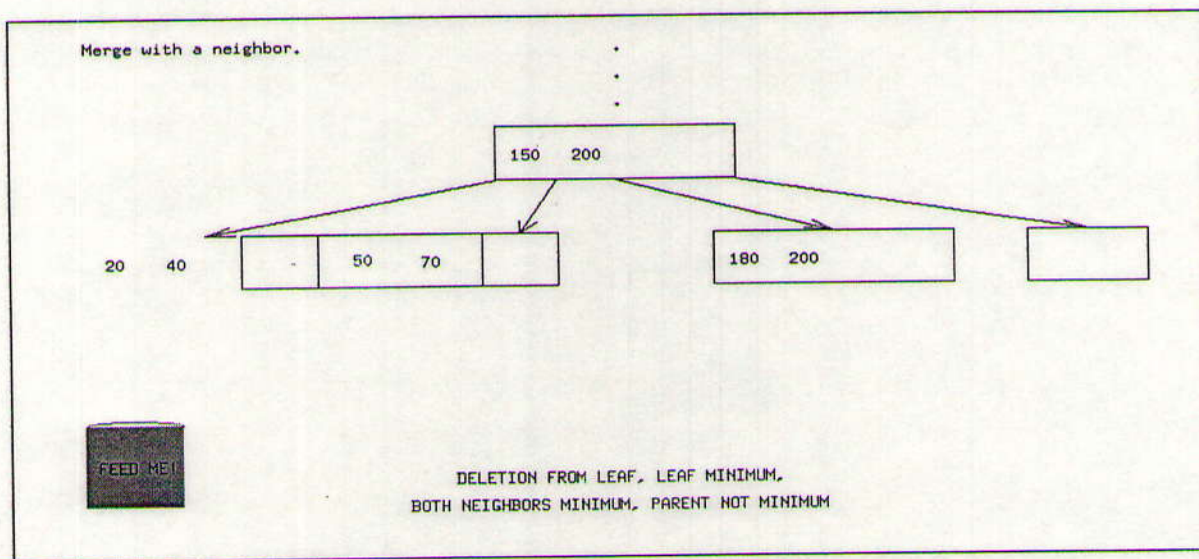


Figure 21. The leaf and its neighbor merge

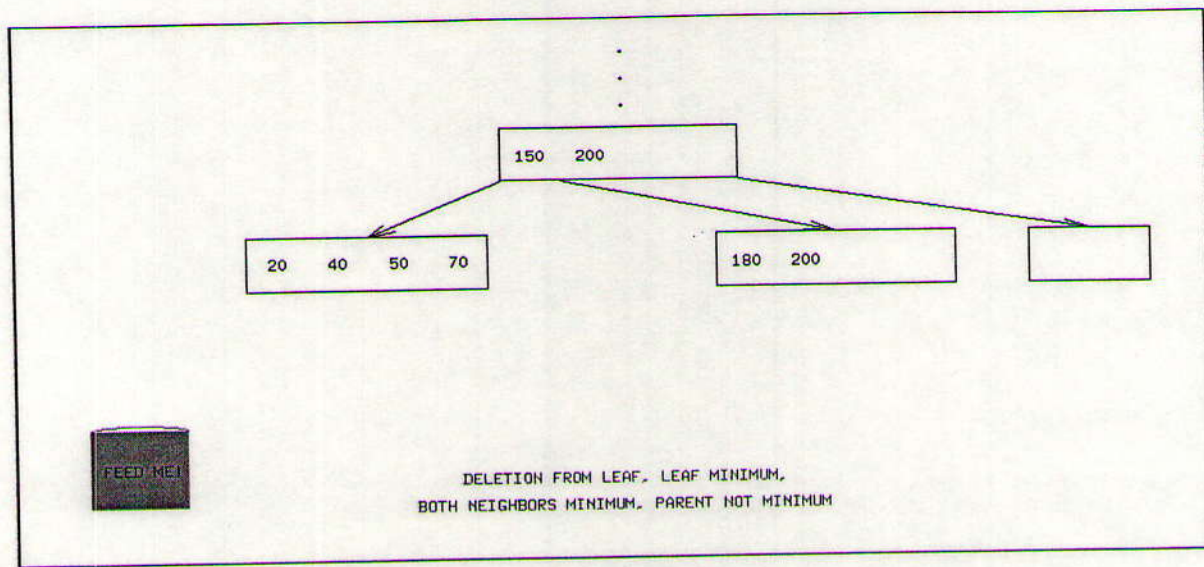


Figure 22. The merging nodes, their links, and values are repositioned as one.

Appendix B

THREADS

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.

Currently, experiments are planned for the following applications:

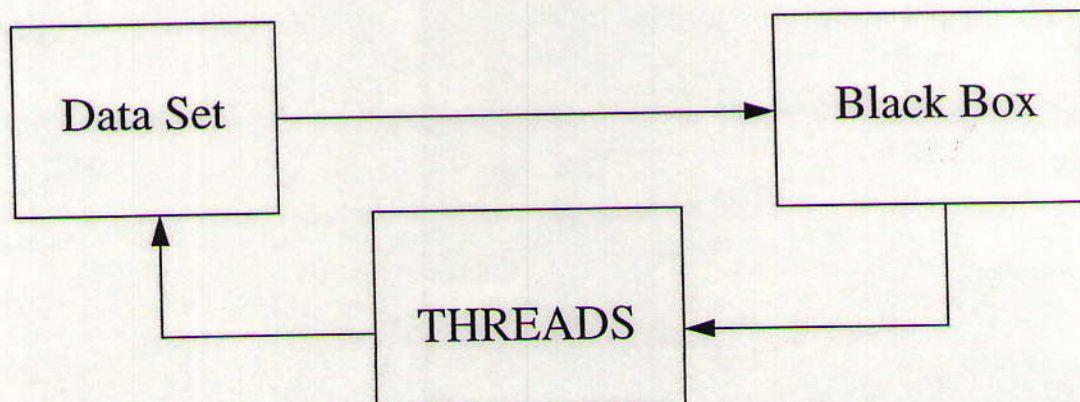
- 1) Big Oh Experiments
- 2) Comparison of different sorting algorithms
- 3) Binary search tree vs. AVL tree
- 4) Hash collision handling
- 5) Big Oh coefficient evaluation

In order to run these types of experiments in a laboratory setting, an appropriate tool is needed. The major part of our project consists of the development of such a tool, named THREADS.

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.

How the Student Uses THREADS

The basic idea behind THREADS is illustrated by the following chart



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 23. 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 generate 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.

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 supplying the write path, a notice prompt

will appear and notify the user to supply THREADS with the appropriate information.

The screenshot shows a window titled "THREADS". Inside, there are several controls: a "Method:" dropdown menu set to "Sort 1"; a "Write to File:" checkbox; a "Write Path:" text field; a "Use File:" text field; a "Sample Size:" spinner box set to "100"; a "Sample Distribution:" dropdown menu set to "Uniform" with a "Distribution Parameters" button next to it; and a "Sample Order:" slider ranging from "0" to "100" with a "-100" label on the left. Below these are buttons for "Run Experiment", "View Graph", "Help", "Clear Table", "Coefficients", and "Quit". At the bottom, there is a section titled "Table of Measurements" with labels "X: Sample Size" and "Y: Measurement" followed by dashed lines for data entry.

Figure 23

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.

- 1) Uniform.
- 2) Exponential.
- 3) Normal
- 4) Gamma
- 5) Poisson
- 6) 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 24).

The screenshot shows a window titled "Distribution Parameters". It contains the following fields:

- 1. Exponential: Mean: 0
- 2. Normal: Mean: 0, Standard Deviation: 5000
- 3. Gamma: Order (a): 5
- 4. Poisson: Mean: 0
- 5. Binomial: Probability %: 50, n: 100

A "Done" button is located at the bottom center of the window.

(Figure 24)

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 '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.

Coefficients

If the 'Coefficients' button is clicked, a small window with five buttons will appear (Figure 25). The five buttons are $\log(n)$, n , $n\log(n)$, n^2 , and n to some power, 'n' being the sample size. If one of these buttons is clicked, an xterm 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 xterm 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.

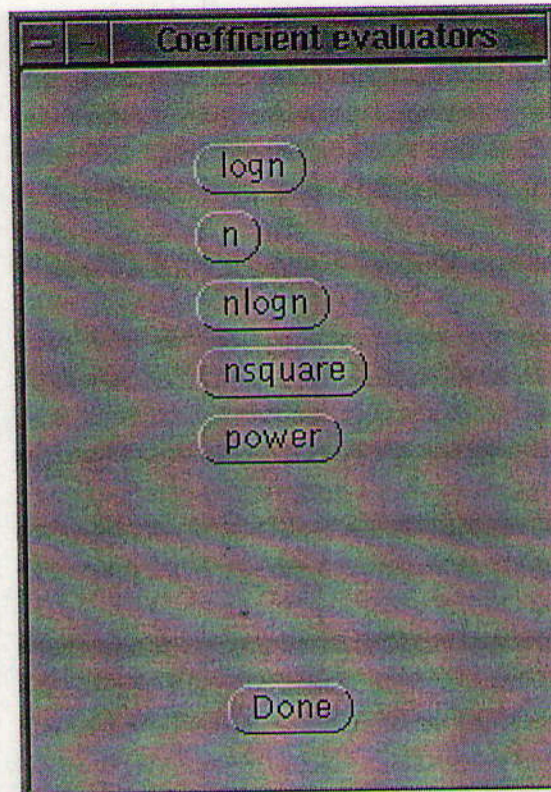


Figure 25

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.

For the Instructor

THREADS is composed of 11 files and a Makefile. The files and their contents are as follows:

threads.G:	Created by the graphical interface code generator, DevGuide 3.0.1.
threads_ui.h:	Definitions of labels used to receive information about the UIT objects.
threads_ui.cc	Sets up the interface, and begins waiting for events to process.
threads_stubs.cc:	Callback functions for the various widgets on the interface. Also contains any auxiliary functions needed by THREADS.
threads.info:	Contains the help text retrieved by pressing the "Help" key on the keyboard.
longhelp.info:	Contains the complete help text displayed in the THREADS help window.
threads.icon:	Contains the graphical data for the THREADS icon.
threads.mask	Contains the graphical data for the THREADS icon mask.
xvgr.prefs:	Contains the preferences for xvgr.
coef.scpt:	Contains the script that is called when one the coefficient buttons are clicked.
arret.cc:	A very small program that just waits until the user presses the Return key; used in coef.scpt.

Black boxes may be added to THREADS with little effort. The spots where the code needs to be modified are marked by comments of the form:

```
/* NEW METHODS: add any new methods here */
```

First, in the file threads_ui.cc, insert the line:

```
(void) mthdchoice.addChoice ("Method name");
```

where 'Method name' is the name to appear on the menu in the interface. For each new method added, a similar line must be inserted. The lines need to be added to the current listing of choices, which is marked in the code.

Next, in the file threads_stubs.cc, add another "else if" condition of the form:

```
else if (strcmp(method, "MethodFile") == 0) {  
    strcat(command, "MethodName");  
}
```

In this section 'MethodFile' is the executable file name for the black box. Each new method needs to have its own case in the "if-else if" statement.

New methods can also be used by placing them in the blackbox directory and typing the name of the method executable in the "Method:" text field. This is much easier from a programming aspect, but this requires the user to type in the name instead of being able to select it from the menu.

New types of distributions may be added in a similar fashion. The menu choices for "Sample Distri-

bution” are added the same way as those for “Method.” A clear definition of what the distribution means and how it will affect random generation of integers will facilitate the changes in the code of `threads_stubs.cc`.

In `threads_stubs.cc`, the system calls use the full path names to execute the black box and to use the other include files. Check that these paths are correct, and change them as appropriate.

Appendix C

Data Structures Lab Manual

Data Structures

Laboratory 1 - Writing an Ada program

1. Write an Ada program to accept as input an integer n and to calculate and print the n th power of two (2^n). Store your program in the file `LAB1_1.ADA`.

Run your program for some values of n and verify that it is correct. List the values and results below:

Find the largest integer value of n for which this program gives a correct value. What is this value of n ?

What happens when you enter a value larger than the above value?

How many bits do you need to represent the largest value of 2^n that Ada can represent?
Hint: it takes k bits to represent $2^k - 1$, and $k+1$ bits to represent 2^k .

Do you think Ada can represent integers that are larger than the largest possible power of 2? If you are not sure, write a little test program to see. If so, what do you think would be the largest representable integer?

Since your computer also has to represent negative integers, and assuming it represents approximately as many negatives as positives, how many bits does your machine use to represent integer?

2. Modify the program you wrote for #1 to express the answer as float instead of integer

type. Store your modified program in the file LAB1_2.ADA.

Run your program for some values of n and verify that it is correct. List the values and results below:

Find the largest integer value of n for which this program gives a correct value. What is this value of n ?

What happens when you enter a value larger than the above value?

The computer represent a floating point number in two parts, the mantissa and the exponent. From your results above, how many digits do you think your computer uses to represent the exponent? Remember that negative exponents are possible as well.

3. Write a program to use Newton's method to calculate $\sqrt{2}$. Newton's method starts with an initial guess, x_0 , and then generates successive guesses using $x_{n+1} = \left(x_n - \frac{2}{x_n} \right)$.

Consider that the process has converged whenever two consecutive guesses are the same or when $x^2 = 2$. For each guess calculate its error as $x^2 - 2$.

How accurate is the answer which you obtain? To how many places is it accurate?

How does your answer here relate to your answer to #2?

Data Structures

Laboratory 2 - Using Ada Packages

In this laboratory, you will solve the same two problems you solved in Laboratory 1, but you will make use of Ada packages to enhance your solutions. The two Ada packages that you will use are described below:

Package `big_integer` is a package which permits you to do arithmetic on integers with infinite precision. You are therefore not limited to the size of integers implemented on the machine that you are using. A full set of arithmetic operators are implemented in this package for data type `big_int`. The specification of the package is

```
-- big_integer.ads

-- Note: It is recommended that this package be used with an Ada
-- implementation that uses automatic garbage collection.

package big_integer is
  type big_int is private;

  procedure put(a : big_int);
  procedure get(a : out big_int);

  function mbi(s : string) return big_int;
  function mbi(i : integer := 0) return big_int;
  -- mbi stands for "Make Big Integer"
  procedure dbi(a : in out big_int);
  function bi2int(a : big_int) return integer;

  function "+"(a, b : big_int) return big_int;
  function "+"(a : big_int) return big_int;
  function "-"(a, b : big_int) return big_int;
  function "-"(a : big_int) return big_int;
  function "*" (a, b : big_int) return big_int;
  function "/"(a, b : big_int) return big_int;
  function "rem"(a, b : big_int) return big_int;
  procedure div(a, b : big_int; result,
    remainder : in out big_int);
  function "abs"(a : big_int) return big_int;

  function "<"(a, b : big_int) return boolean;
  function "<=" (a, b : big_int) return boolean;
  function ">"(a, b : big_int) return boolean;
```



```

function ">=" (a, b : big_int) return boolean;
function equal(a, b : big_int) return boolean;
    -- This is not the same as the operator "=".
    -- When making arithmetic comparisons for equality, use
-- the above function.
    -- Use "=" only if you know what you are doing!
function equal(a : big_int; b : integer) return boolean;
function asfloat(a : big_int) return float;
function "mod" (a, b : big_int) return big_int;
big_int_error : exception;

private
    type digit_node;
    type dn_ptr is access digit_node;
    type digit_node is
        record
            d : integer range 0..9;
            next : dn_ptr;
        end record;

    type big_int is
        record
            is_pos : boolean;
            dl : dn_ptr;
        end record;
end big_integer;

```

Below is an example program which uses the type bigint. This program should serve as a pattern for the programs you need to write in this and the following laboratories:

```

-- Program to compute the square of an integer and store and print as a bigint.

with text_io; use text_io;
package int_io is new integer_io(integer);
with int_io; use int_io;
with text_io; use text_io;
with big_integer; use big_integer;

procedure lab2_example is

    square : big_int := mbi(1);
    number : integer;

begin
    get(number);
    square := mbi(number)*mbi(number); -- mbi converts integer to bigint.
    put(square);                       -- this calls put from package big_integer.
end lab2_example;

```


1. Rewrite the program you used in Laboratory 1 to calculate powers of 2, using type `big_int` instead of integer for the calculations. Calculate 2 to the powers 1..16, 1023, 1024, and 2048.

Verify that the first 16 powers of 2 are correct from the answers obtained in Laboratory 1.

What are some ways that you can spot check the answers to 2^{1023} and 2^{1024} ?

How many digits are there in 2^{1024} ? How many in 2^{2048} ? How many would you expect in 2^{4096} ?

Another package that is provided is one that does calculations for rational numbers. A rational number is a number that is represented by two integers, a numerator and a denominator. For example, the float number 0.5 would be represented by the two integers (1,2) since it is equal to $\frac{1}{2}$.

The specification of the package `rat_pack` defining type rational is

Package `Rat_Pack` is

```
type Rational is private;

function "+" (R1,R2:Rational) return Rational;
function "-" (R1,R2:Rational) return Rational;
function "*" (R1,R2:Rational) return Rational;
function "/" (R1,R2:Rational) return Rational;
function Rat(I1,I2:integer) return Rational;
    -- converts two integers to a rational
function Numer(R:Rational) return integer;
    -- returns the numerator of a rational number
function Denom(R:Rational) return integer;
    -- returns the denominator of a rational number
function "<" (R1,R2:Rational) return boolean;
function ">" (R1,R2:Rational) return boolean;
procedure Put(R:Rational);
    -- outputs a rational number
function Value(R:Rational) return float;
    -- returns a float approximation to a rational number

Zero_Denominator : exception;
```



```
-- raised when a rational operator results in a zero
-- denominator
```

```
private
  type Rational is record
    Num : integer;
    Den : integer;
  end record;
end Rat_Pack;
```

2. Using the package rat_pack, repeat the calculation of the square root of two using Newton's method.

What is the best approximation, measured by the number of digits in the denominator, that can be generated using type rational?

```
package big_rational is
```

How does this compare to the best approximation possible using float?

```
function "+"(r1, r2 : big_rat) return big_rat;
function "-"(r1, r2 : big_rat) return big_rat;
function "*" (r1, r2 : big_rat) return big_rat;
function "/" (r1, r2 : big_rat) return big_rat;
function "**"(r1, r2 : big_rat) return big_rat;
function "/"(r1 : big_rat; i1 : integer) return big_rat;
function rat(i1, i2 : big_int) return big_rat;
function rat(i1, i2 : integer) return big_rat;
function sqrt(r : big_rat) return big_rat;
function denom(r : big_rat) return big_int;
function "<"(r1, r2 : big_rat) return boolean;
function ">"(r1, r2 : big_rat) return boolean;
procedure put(r : big_rat);
function value(r : big_rat) return float;
```

Why was this exception raised by your program at this time?

```
zero_denominator : exception;
```

```
private
  type big_rat is
    record
      num : big_int;
      den : big_int;
    end record;
end big_rational;
```

1. Using this package and type big_rat, recalculate your estimates of the square root of two. Ter-

minate your calculation after the seventh estimate.

What is the accuracy that you obtained on the seventh estimate?

Next we introduce the following generic package:

```
-- ratgen.ads
--
-- This is a generic package to define rational numbers over the
--   type inttype.
-- Multiple operators on inttype are passed as parameters.

generic
  type inttype is private;
  with function "+"(i1,i2 : inttype) return inttype;
  with function "-"(i1,i2 : inttype) return inttype;
  with function "-"(i1 : inttype) return inttype;
  with function "*" (i1,i2 : inttype) return inttype;
  with function "/" (i1,i2 : inttype) return inttype;
  with function "rem"(i1,i2 : inttype) return inttype;
  with function equal(i1,i2 : inttype) return boolean;
  with function unequal(i1 : inttype; i2 : integer) return
boolean;
  with function "<"(i1,i2 : inttype) return boolean;
  with procedure put(i1 : inttype; width : integer := 1000;
base :
                                integer := 10);
  with function asfloat(i1 : inttype) return float;
package rat_gen is
  type rational is private;

  function "+"(r1, r2 : rational) return rational;
    -- sum of two rationals
  function "+"(r1 : rational) return rational;
    -- identity operator
  function "+"(r1 : rational; i1 : inttype) return rational;
    -- adds a rational to an inttype
  function "+"(i1 : inttype; r1 : rational) return rational;
    -- adds an inttype to a rational
  function "-"(r1, r2 : rational) return rational;
    -- difference of two rationals
  function "-"(r1 : rational) return rational;
    -- negation operator
  function "-"(r1 : rational; i1 : inttype) return rational;
    -- subtracts inttype from rational
  function "-"(i1 : inttype; r1 : rational) return rational;
```



```

    -- subtracts rational from inttype
function "*" (r1, r2 : rational) return rational;
    -- product of two rationals
function "*" (r1 : rational; i1 : inttype) return rational;
    -- multiplies rational by inttype
function "*" (i1 : inttype; r1 : rational) return rational;
    -- multiplies inttype by rational
function "/" (r1, r2 : rational) return rational;
    -- quotient of two rationals
function "/" (r1 : rational; i1 : inttype) return rational;
    -- divides rational by inttype
function "/" (i1 : inttype; r1 : rational) return rational;
    -- divides inttype by rational
function rat (i1, i2 : inttype) return rational;
    -- creates rational i1/i2 out of two inttypes
function numer (r : rational) return inttype;
    -- returns the numerator of the rational r
function denom (r : rational) return inttype;
    -- returns the denominator of the rational r
function "<" (r1, r2 : rational) return boolean;
    -- less than comparison for two rationals
function ">" (r1, r2 : rational) return boolean;
    -- greater than comparison for two rationals
procedure put (r : rational);
    -- output function for rational
function value (r : rational) return float;
    -- converts rational to real

zero_denominator : exception;

private
    type rational is
        record
            num : inttype;
            den : inttype;
        end record;
end rat_gen;

```

2. Using `rat_gen` with `inttype` defined as integer, rerun your program for the square root of 2 estimation.

Why did this fail to obtain the accuracy that you obtained in #1?

What caused the process to fail.

3. Finally, apply the type `big_integer` to the generic type `rat_gen` to obtain rationals over the

`big_integer` type. Run your square root of two program using this data type and observe the results. Once again, terminate your program after the seventh estimate.

How does the seventh estimate compare with the seventh estimate you obtained in #1?

What advantage do you see to using this generic approach over the approach of using a simple type in #1?

Data Structures

Laboratory 4 - Big Oh Sampling

Threads is a tool for running experimental programs and examining measurements reported from those programs. This laboratory is your first experience using Threads software.

Read the Threads help file and familiarize yourself with the capabilities of this software.

Five programs are provided for you to observe the behavior of various big-oh rates using Threads.

1. Lab4_1 is a program which computes 2^n using an algorithm which is $O(\log n)$. Run this program through Threads for $n=64, 128, 256, 512$ and graph the resulting data. For each value of n , run the experiment three times to generate three data points at each value of n .

Compare the graph of your data with the graph of the logarithm function. The best way to do this is to plot this graph using the Log-linear axes. Compare the resulting graph to a straight line. The nearer this approximates a straight line, the nearer your times are to $O(\log n)$. Record below the data table generated and your observations concerning the graph.

2. Lab4_2 is a program which sums the first n integers. It should run with a time of $O(n)$. Run it through Threads with three repetitions for $n=50, 100, 150, 200, 250$, and print your table and graph.

Record your data table in the space below:

Your graph should approximate a straight line if the time of this algorithm is actually $O(n)$. Does it appear to approximate a straight line?

3. Lab4_3 is a program which calculates 2^i for $i=1, \dots, n$. It should have a running time of $O(n \log n)$. Run this program through Threads with three repetitions for $n= 50, 100, 150, 200, 250, 300$, and record your table of results below:

Graph the table of values generated. Change your graph of the above data to plot on Log-log axes. If this is truly $O(n \log n)$ time, this graph should appear as a straight line. Is this the case?

4. Lab4_4 is a program which calculates a multiplication table for all pairs of integers in $1..n$. It should run with a time of $O(n^2)$. Run it through Threads with three repetitions for $n=50, 100, 150, 200$ and copy your table values below:

Change your graph of the above data to plot on Log-log axes. If this is truly $O(n^2)$ time, this graph should appear as a straight line with slope two. Is this the case? How do you estimate the slope?

5. Lab4_5 is a program which recursively calculates 2^n via repeated additions. It does this very poorly from a big-oh point of view and runs with a time of $O(2^n)$. Because this algorithm has a run-time which grows very rapidly, run through Threads with three repetitions for $n=10, 11, 12, 13$ and 14. Notice that the run-time approximately doubles every time n is increased by 1. Record your table values below:

Change your graph of the above data to plot on Linear-log axes. If this is truly $O(2^n)$ time, this graph should appear as a straight line. Is this the case?

6. Answer the following questions after reflecting on the results of the five tests that you have run.

How accurately did the results you obtained agree to the results that you expected? In which cases was the agreement the greatest and which cases the least?

What explanations might there be for the discrepancy between the expected results and the observed results in these experiments?

How do you explain the variance, if any, in the times reported when the same experiment is repeated several times?

Data Structures

Laboratory 5 - Big Oh Determination

This laboratory requires you to analyze 10 different test programs to determine the big-oh of the running time. You will first be asked to determine the running time by analyzing the code for the function and then you are to verify your hypothesis by using Threads. A useful fact for working with logarithms in this lab is the following identity that can be used to convert logs from base a to base b:

$$\log_a n = \frac{\log_b n}{\log_b a}$$

```
function Test1(n:integer) return integer is
count : integer:=0;
newn   : integer:=n/2;
begin
  while newn>0 loop
    count:=count+1;
    newn:=newn/2;
  end loop;
  return count;
end Test1;
```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary. Hint: try n=1,2,3,4,5,6,7,8 and see if any pattern emerges that is helpful.


```

function Test2(n:integer) return integer is
count : integer:=0;
newn  : integer:=n;
begin
    while newn>0 loop
        count:=count+1;
        newn:=newn-1;
    end loop;
    return count;
end Test2;

```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```

function Test3(n:integer) return integer is

```



```

count : integer:=0;
begin
  for i in 1..n loop
    count:=count+Test1(n);
  end loop;
  return count;
end Test3;

```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```

function Test4(n:integer) return integer is
count : integer:=0;

```



```
begin
  for i in 1..n loop
    count:=count+Test2(n);
  end loop;
  return count;
end Test4;
```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```
function Test5(n:integer) return integer is
count : integer:=0;
begin
```



```

for i in 1..n loop
    count:=count+Test4(n);
end loop;
return count;
end Test5;

```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```

function Test6(n:integer) return integer is
count : integer:=0;
newn  : integer:=n/2;
begin

```



```

while newn>0 loop
  count:=count+test1(n);
  newn:=newn/2;
end loop;
return count;
end Test6;

```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. Caution: don't try to run this with n greater than 20. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```

function Test7(n:integer) return integer is
count : integer:=0;
begin
  if n>0 then
    count:=count+test7(n-1)+1;

```



```
else
    count:=0;
end if;
return count;
end Test7;
```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```
function Test8(n:integer) return integer is
count : integer:=0;
newn  : integer:=n;
begin
    if n>0 then
        count:=count+test8(n-1)+test8(n-1);
```



```

else
    count:=1;
end if;
return count;
end Test8;

```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

```

function Test9(n:integer) return integer is
count : integer:=0;
begin
    if n>1 then
        count:=count+test9(n/2)+1;
    else
        count:=0;
    end if;
end Test9;

```



```
    end if;  
    return count;  
end Test9;
```

What is the running time for this function in big-oh notation? Justify your answer.

Show results from running this function through Threads. How does this verify or contradict your previous answer? Change your hypothesis if necessary

Data Structures

Laboratory 6 - Stacks and Queues

For this laboratory you will be introduced to algorithm animation. In order to do this we will use the special software package called XTango.

1. Run XTango for the Post Office simulation. In order to do this you need to type the command

```
/home/csci286/xtango/anim/post
```

In this case, we simulate a post office with n serving stations, where n is an input to the program. A simulation is then run and you will be asked to observe the total waiting time for the 25 customers that are served. The file that contains the arrival times that you will use is

```
/home/csci286/po.dat.
```

Record the total waiting times in the table below:

Number of Stations	Total Wait Time
1	_____
2	_____
3	_____
4	_____

What is the limit in the size of queue allowed in this simulation? In other words, after the queue reaches a certain size, customers no longer enter the system. What is that size?

Did you notice any inefficiencies in the way the queueing systems works? What could be done to make the queues operate more efficiently, that is, with less total wait time?

3. The Ada program for converting a string from infix to postfix form is given below:

```
while not end_of_input loop
  get(next_symbol);
  if operand(next_symbol) then
    put(next_symbol);
  elsif next_symbol=')' then
    while top(stack /= '(' loop
      put(pop(stack));
    end loop;
    pop(stack);
  else
    while not Empty(stack) and then
      Priority(next_symbol) <= Priority(top(stack)) loop
      if top(stack) /= '(' then
        put(pop(stack));
      end if ;
    end loop;
    push(next_symbol);
  end if;
end loop;
while not empty(stack) loop
  put(pop(stack));
end loop;
```

The priorities used for the operands in this algorithm are given in the following table:

Operand	Priority
(2
* /	1
+ -	0

In this algorithm, get means get the next character from the input stream and put means put the next character to the output stream. Push and pop refer to operations on a separate stack.

Run the XTango animation for this program by typing

```
/home/csci286/xtango/anim/postfix
```

Run this for the input stream a+b to see how the algorithm operates.

To see how parentheses are handled, run XTango for the input stream a+(b*c).

To see how priority is used, run XTango for the input streams a+b*c and a*b+c.

Make up a complex expression that contains at least 5 operators and uses both parentheses and priority. Write your expression in the space below:

Run through the algorithm by hand and predict the resulting output stream below:

Now run the algorithm in XTango and verify your result.

Data Structures

Laboratory 7 - Comparison of AVL and BST

For this laboratory, you will be using THREADS to compare the use of AVL trees and Binary Search trees without balancing. Two programs are provided in the Method menu of THREADS which you will use for this lab. They are TestAVL and TestBST. They return the total depth of all of the nodes in a binary search tree with X nodes generated at random.

1. Run TestBST to generate binary search trees for sample sizes 100,200,400,800,1600,3200. Run the experiment three times for each sample size.

(a) Enter the values you obtained below:

(b) Make a graph of the values you obtained. Does the graph look linear?

(c) Now we will perform a more sophisticated test for linearity, using the "Coefficients" button on THREADS. Generate the coefficients for n, $n \log n$, and power. What conclusions do you draw from your results as to the growth rate of the total depth of all nodes in a random binary search tree?

2. Now repeat the above experiment for TestAVL. First clear out your table in THREADS, then rerun all of the experiments.

(a) Write your results below:

(b) How do these values compare with the values you obtained for TestBST? How do you explain this comparison?

(c) Run your data through "Coefficient" again for n , $n \log n$, and power. How does this compare with what you observed for TestBST?

3. Now we're going to place the entries in the tree in an ordered fashion. That is, we are going to insert into the tree from smallest to largest. You set the order by changing "Sample Order" in THREADS to 100. (-100 would do the same, but place the values in descending order.) Do this and rerun the experiments for TestAvl.

(a) Enter the results below:

(b) Use coefficients to observe the growth rate of this data. What do you observe?

4. Now repeat the same process for TestBST. This time, limit your sample sizes to 100,200,400, and 800.

(a) Report your results below:

(b) Use Coefficients to observe the growth rate. What do you observe? How do you explain this?

(c) Why did I not have you run this experiment for sample sizes greater than 800?

5. Now try the same experiment with the sample only partially ordered. You can do this if you set Sample Order to 20. This means that the first 20% of the sample is in order and the remainder is random. Now rerun the experiment in 4 using this ordering.

(a) Report your results below:

(b) Use Coefficients to estimate the growth rate. What do you see and what is your estimate?

(c) What have you learned from this entire laboratory?

Data Structures

Laboratory 8 - Sort Comparisons

For this laboratory, you will be using THREADS to compare various sorting methods. The five sort methods that we will compare are Merge Sort, Shell Sort, Heap Sort, Quicksort, and Insertions Sort.

1. Run Sort1, Merge Sort, and observe the elapsed time for sorting 1,000 to 10,000 values in steps of 1,000.

(a) Enter the time for 10,000 below:

(b) This sort is Merge Sort which is supposed to be an $n \log n$ sort. Use the Coefficient button of Threads to test this hypothesis. Are the values you obtain fairly stable? Do you conclude that your observations are $n \log n$?

(c) Now rerun the sorts for sample size 10,000, but use ordered data. What was the time that you observed for 10,000? Is this an improvement over the time observed in (a)?

(d) Finally, run Sort1 for 10,000 sample size with the values in reverse order (Sample Order = -100). How does this compare to the times observed in (a) and (c)?

2. Run Sort2, Shell Sort, and observe the elapsed time for sorting 1,000 to 10,000 values in steps of 1,000.

(a) Enter the time for 10,000 below:

(b) This sort is Shell Sort which has an undetermined big Oh. Use the Coefficient button of Threads to make an hypothesis about its big Oh. What would be your best estimate?

(c) Now rerun the sorts for sample size 10,000, but use ordered data. What was the time that you observed for 10,000? Is this an improvement over the time observed in (a)?

(d) Finally, run Sort2 for 10,000 sample size with the values in reverse order (Sample Order = -100). How does this compare to the times observed in (a) and (c)?

3. Run Sort3, Heap Sort, and observe the elapsed time for sorting 1,000 to 10,000 values in steps of 1,000.

(a) Enter the time for 10,000 below:

(b) This sort is Heap Sort which is supposed to be an $n \log n$ sort. Use the Coefficient button of Threads to test this hypothesis. Are the values you obtain fairly stable? Do you conclude that your observations are $n \log n$?

(c) Now rerun the sorts for sample size 10,000, but use ordered data. What was the time that you observed for 10,000? Is this an improvement over the time observed in (a)?

(d) Finally, run Sort3 for 10,000 sample size with the values in reverse order (Sample Order = -100). How does this compare to the times observed in (a) and (c)?

4. Run Sort4 Quicksort, and observe the elapsed time for sorting 1,000 to 10,000 values in steps of 1,000.

(a) Enter the time for 10,000 below:

(b) This sort is Quicksort which is supposed to be an $n \log n$ sort. Use the Coefficient button of Threads to test this hypothesis. Are the values you obtain fairly stable? Do you conclude that your observations are $n \log n$?

(c) Now rerun the sorts for sample size 10,000, but use ordered data. What was the time that you observed for 10,000? Is this an improvement over the time observed in (a)?

(d) Finally, run Sort4 for 10,000 sample size with the values in reverse order (Sample Order = -100). How does this compare to the times observed in (a) and (c)?

5. Run Sort5, Insertion Sort, and observe the elapsed time for sorting 1,000 to 5,000 values in steps of 1,000. The reason that we use 5,000 instead of 10,000 will become obvious.

(a) Enter the time for 5,000 below:

(b) This sort is Insertion Sort which is supposed to be an n^2 sort. Use the Coefficient button of Threads to test this hypothesis. Are the values you obtain fairly stable? Do you conclude that your observations are n^2 ?

(c) Now rerun the sorts for sample sizes 1,000 to 10,000, but use ordered data. What was the time that you observed for 5,000? Is this an improvement over the time observed in (a)?

(d) Finally, run Sort3 for 5,000 sample size with the values in reverse order (Sample Order = -100). How does this compare to the times observed in (a) and (c)? Do you have any explanation for this?