Matrix — Algorithm simulation tool

June 22, 2004

Contents

1 Introduction **3**

2 System requirements **3**

2.1 Platforms 3

2.1.1 Linux 3

2.1.2 Windows 4

2.1.3 MacOS 4

3 Setting Up the System **4**

3.1 Compiling GNU Jaxp 5

3.2 Configuring the system 5

4 How to Run Matrix **5**

4.1 Command line arguments 6

5 Interaction **7**

5.1 Mouse actions 7

5.2 Hotspots 7

5.3 Data Structure Manipulation 8

5.3.1 Inserting Keys into a Data structure 9

5.3.2 Deleting Keys and Nodes from a Data Structure 10

5.3.3 Other Operations 11

5.4 Animation 11
5.4.1 Control panel 12
5.5 Customizing Visualization 12

6 Menu Commands **13**
6.1 File Menu 13
6.2 Format Menu 15
6.3 Insert Menu 15
6.4 Options Menu 16
6.5 Animator Menu 17
6.6 Content Menu 18
6.7 Pop-up Menu 18
6.7.1 Visualization Menu 19
6.7.2 Filters Menu 21

7 Adding New Data Structures **22**

8 FAQ **22**

A APPENDIX - Text file formats **24**

1 Introduction

Matrix is a portable algorithm animation and simulation framework developed in Laboratory of Information Processing Science, (Department of Computer Science and Engineering) at the Helsinki University of Technology. The system allows direct manipulation of basic data structures like arrays, linked lists, trees and graphs and their composites, as well as a number of abstract data types, such as Binary Search Trees. It stores the manipulation process thus allowing the user to create algorithm animations without writing any code. In addition, it is possible to visualize, animate and simulate one's own code.

The system is roughly divided into three different parts. There is the Matrix Framework, which is the hearth of the system. The framework contains all functionality required to create applications that can be used for visualization, animation and simulation of data structures and algorithms.

There is also a prototype application, which is used in the development of the system. This application contains all functionality that is working well-enough to be used.

The prototype application is not the only application that uses the Framework. The TRAKLA2 system, which is a web-based system for solving data structures and algorithms -related exercises, also uses the Matrix Framework.

2 System requirements

Use javac (Java Compiler) 1.2 or newer to compile the code. A graphical user interface (Windows, X, etc.) with java run-time environment is required to run the prototype.

A SAX (Simple API for XML) library is required. Java 1.4 includes a SAX library. We have included GNU Jaxp, an XML library created by GNU project, in the Matrix release. If you have another SAX library, such as Crimson by Apache, or use Java 1.4, you do not need GNU Jaxp.

2.1 Platforms

Java should be platform-independent. Unfortunately the implementations on different platforms are not completely identical. We have tested the system on several different platforms, and encountered some problems. In the following we have a list of tested platforms together with platform-specific notes.

Most of the development of Matrix is done in Linux or MacOS X environment. Therefore most of the system testing has been done in those environments.

2.1.1 Linux

Matrix runs successfully on Linux using java versions 1.2, 1.3 and 1.4. The implementation of pop-up menus in Linux Java is, however, buggy. You cannot select items

in pop-up menus in the normal way of merely selecting an item while the right menu button is pressed, and releasing the button when the cursor is over the correct item. In Linux Java you must select the desired menu item using the left mouse button. However, in the submenus, the normal way of using the right mouse button works.

2.1.2 Windows

Matrix has successfully been run on Windows 98, Windows 2000 and Windows XP using Java 1.2 or newer. There have been a few problems with the pop-up menus on Windows as well. See FAQ (Section 8) for possible solutions to pop-up menu problems. On older Windows versions the look and feel can be a little bit peculiar.

2.1.3 MacOS

Matrix has successfully been run on MacOS X using Java 1.2. No Mac-specific issues have been found.

3 Setting Up the System

Unpack the Matrix distribution into a directory of your choice. In this discussion "$MATRIX" refers to the directory Matrix was unpacked. The actual directory may vary between versions. For example, if you unpacked the Matrix source code distribution version 4 in the directory "/home/login/", the distribution is now in directory "/home/login/matrix-4.0/". Within the jar distribution, however, some files and directories have been omitted, and the distribution does not unpack itself into a subdirectory. Moreover, if you're running Windows, you should replace slashes ("/") with backslashes ("\").

To compile Matrix using Apache Ant (http://ant.apache.org/) you can use "`ant compile`" or just "`ant`" in the $MATRIX/code directory.

The Matrix distribution has Makefiles for Unix systems. Try "`gmake`" and/or "`make`" in the $MATRIX/code directory. (The Makefiles might not always work, because different Java implementations for Unix seem to have a different command-line syntax for some reason.)

If that does not work, you can try to use the command
"`find . -name '*.java' | xargs javac`" to compile the code.

In a Windows system try "`compile.bat`" to compile Matrix source files.

If that doesn't work, you can try to compile the system manually. The command "`javac */*.java */*/*.java */*/*/*.java */*/*/*/*.java`" should work. In any case, the Matrix root directory must be included in the java CLASSPATH in order to the system to compile. If your CLASSPATH includes the current working directory, this should be no problem as long as you are in the $MATRIX/code directory while trying to compile the program.

3.1 Compiling GNU Jaxp

GNU jaxp should be automatically compiled along with the rest of the system. If that does not work you can compile GNU Jaxp by yourself. The GNU Jaxp subdirectory $MATRIX/gnujaxp-1.0beta1 within the release contains a makefile for GNU Jaxp. Compile the library and make a jar file called gnujaxp.jar by"`gmake`" and/or "`make`" in the GNU Jaxp subdirectory. Copy the jar file into the directory $MATRIX/lib, and everything should work. If that fails, consult the GNU Jaxp README for further instructions.

3.2 Configuring the system

The configuration of the prototype application is stored in the matrixconf.xml file, which is stored in the $MATRIX/code directory. The corresponding document type definition, matrixconf.dtd is in the same directory. Because of buggy GNU Jaxp code, the xml configuration file is not validated by default. See section 4.1 and the configuration file for further information.

For a normal user, there should not be any reason to touch the configuration file, since most of it defines the contents various menus as well as the mapping between data structure types and visualizations. Some system defaults are set in the very beginning of the configuration file.

The details of the configuration system are explained in the Matrix Configuration document, located in the directory $MATRIX/docs/.

4 How to Run Matrix

To run Matrix using Ant use the command "`ant run`" in the $MATRIX/code directory.

In Unix or Linux Matrix can be launched using the "`matrix`" startup script or by calling the Java Virtual Machine directly. In Windows environment "`matrix.bat`" can be used to start the program. In both cases make sure that the command java can be found in your PATH and that Matrix classes are in the CLASSPATH. The startup scripts are in the $MATRIX/bin/ directory. In Unix or Linux Matrix can also be launched with the command "`make run`" in the $MATRIX/code/ directory.

The scripts make no changes to your classpath. If you're using Java 1.3 or older the file gnujaxp.jar must be in your classpath. The default location for this file is $MATRIX/lib/. Furthermore, if you move the startup scripts away from $MATRIX/bin directory or call them from other directories, you must modify the scripts. By default, the scripts assume that Matrix configuration file can be found in directory ../code/. Change this relative path to an absolute one.

If you have a jar release of Matrix, you should be able to start it by double clicking in Windows or MacOS. If that doesn't work, the program can be started with command "`java -jar matrix.jar`". The name of the jar file can also be changed to anything. The jar package contains GNU Jaxp, so there is no need to modify classpath.

Matrix can also be started manually by trying the following commands:

"`java prototype.ui.StartFrame`", or

"`java -classpath .:../lib/gnujaxp.jar prototype.ui.StartFrame`"

in the $MATRIX/code/ directory.

When started, Matrix will create an animation window with a menubar and a set of buttons at the center of the window. On Windows XP, Matrix looks as illustrated in Figure 1.

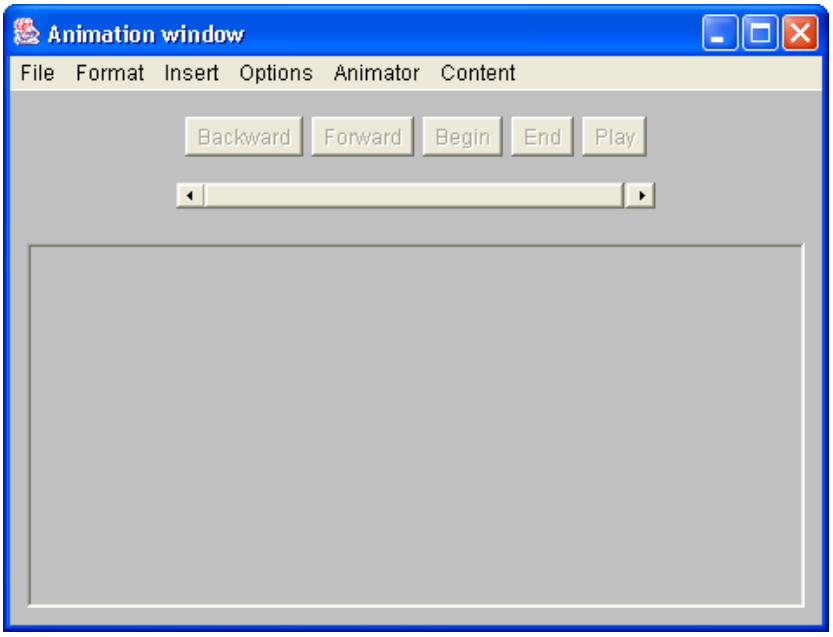

Figure 1: Matrix main window on Windows XP

4.1 Command line arguments

It is possible to give Matrix two optional command line parameters when calling the virtual machine directly (the startup scripts in $MATRIX/bin take no parameters). First parameter is the location of the Matrix configuration file. If the system is told the location of the configuration file, it will search for the file from that location. Otherwise, it will try to find the configuration file in the current working directory.

Matrix can be started with specific configuration file by using a command like "`java prototype.ui.StartFrame /some/path/myconfig.xml`"

If and only if the configuration file has been specified, the system can also be told to validate the xml file using a dtd. This is done by telling Matrix the location of the xml configuration file. However, since several xml parsers (including the one shipped with

Java 1.4) do not validate a document unless explicitly told to do so (in program code, NOT in the xml file) also Matrix must explicitly be told to use validation.

Validation is turned on by giving Matrix a second command line parameter, which must be either string "true" or "t" (case does not matter). Therefore, the system can be told to use both specific configuration file and validation with command "`java prototype.ui.StartFrame /some/path/myconfig.xml true`". The xml configuration file must also contain the correct !DOCTYPE tag.

Validation is turned off by default because GNU Jaxp cannot read relative paths to dtd files correctly. If you wish to use validation with GNU Jaxp, you must give the absolute path to the dtd-file used in <!DOCTYPE> -element of the configuration file.

5 Interaction

Two kinds of functionality are provided for interaction with the system. First, *control over the visualization* is allowed, for example, in order to adjust the amount of detail presented in the display, to navigate through large object graphs, or to control the speed and direction of animations. It should be noted that the system is always visualizing an actual underlying data structure. The display contains the actual run-time topology of an object-oriented program, and provides automatically produced layouts with different levels of details.

Second, some meaningful ways to make experiments are needed in order to explore the behaviour of the underlying structure. Thus, Matrix allows the user to *change the state of the underlying object graph* in terms of direct manipulation. Both kinds of functionalities are needed for exploring the underlying structure.

5.1 Mouse actions

The left mouse button is used to select, drag and drop objects. The right button is used to open the **pop-up menu**. The contents of the pop-up menu depend on the active object.

An object can be selected by left-clicking it. The basic manipulation can be done by dragging and dropping objects. For instance, in order to insert a key from an array into a binary search tree, the desired key must be dragged from the array and dropped onto the tree. When an object is being dragged, a frame appears around it (see Fig. 2).

5.2 Hotspots

All data structures with a visible title have a hotspot (a square containing a red "`[^]`" inside it) in the upper left corner of the visualization. The pop-up menu can be opened by left-clicking on this hotspot.

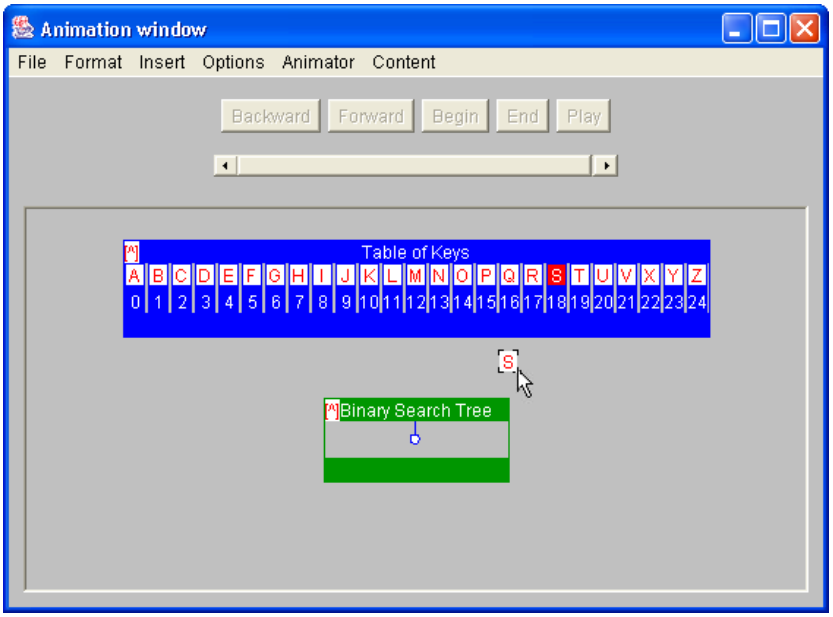

Figure 2: A key from an array can be inserted into a tree by dragging and dropping it onto the tree

Array visualizations contain another hotspot in their upper right corner. An arrow ("->") appears when the mouse cursor is held over this hotspot. The number of positions the array can hold can be changed by dragging this hotspot left or right. Hotspots are illustrated in Figure 3.

Figure 3: The hotspot in the upper left and right corner of an array pops up a menu and resizes the array, respectively

5.3 Data Structure Manipulation

New data structures can be created by using the Insert menu. For more information on the various kinds of data structures, see the section on the Insert menu.

Select a new data structure from the menu and the visualization of the data structure will be shown in the window. If the window contains many other representations, the new data structure may be outside the visible area of the animation window. If the new structure does not appear in the animation window, use the scrollbars or enlarge the window.

The visualized structures can easily be manipulated using mouse actions. Most of the available commands can be accessed through the pop-up menu. The following subsections describe the default behavior for the structures shipped with the Matrix release. However, by programming one's own data structures all the operations can be replaced by new functions. Refer to the Programmer's Manual for more details.

5.3.1 Inserting Keys into a Data structure

Keys can be inserted into a data structure by dragging them from another structure (such as a table of keys) onto the data structure. Matrix will then invoke the insertion routine of the corresponding data structure and update the visualization.

It is also possible to insert a key in a specific node of a data structure. To do so, drag and drop the new key into the desired node.

Remember that inserting a key in a structure (for example, an AVL tree) is not the same as inserting it in a node of the structure (for example, a leaf node of an AVL tree). If you wish to call the insertion routine of an ADT, drop the keys over the title of the visualization (see Fig. 4).

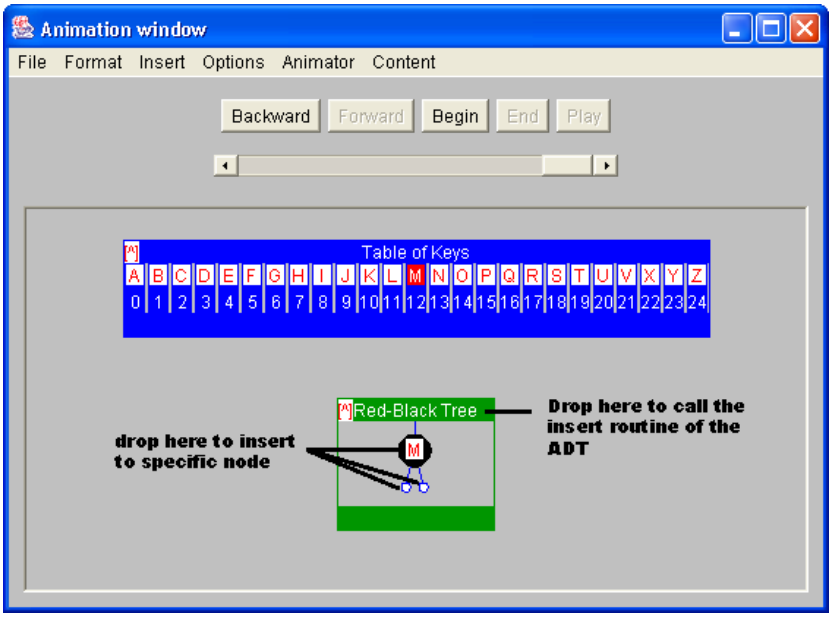

Figure 4: Keys can be inserted into a specific node or by invoking the insert routine for an ADT

It is also possible to insert sets of keys and some other structures into data structures with an insertion routine. For example, in order to insert all the keys in a table of keys into a binary search tree, drag and drop the entire array onto the title bar of the target search tree (see Fig. 5). This will insert the keys one by one. Not all data structures support this functionality. For example, inserting a table into such a structure will cause the table to be inserted as a whole.

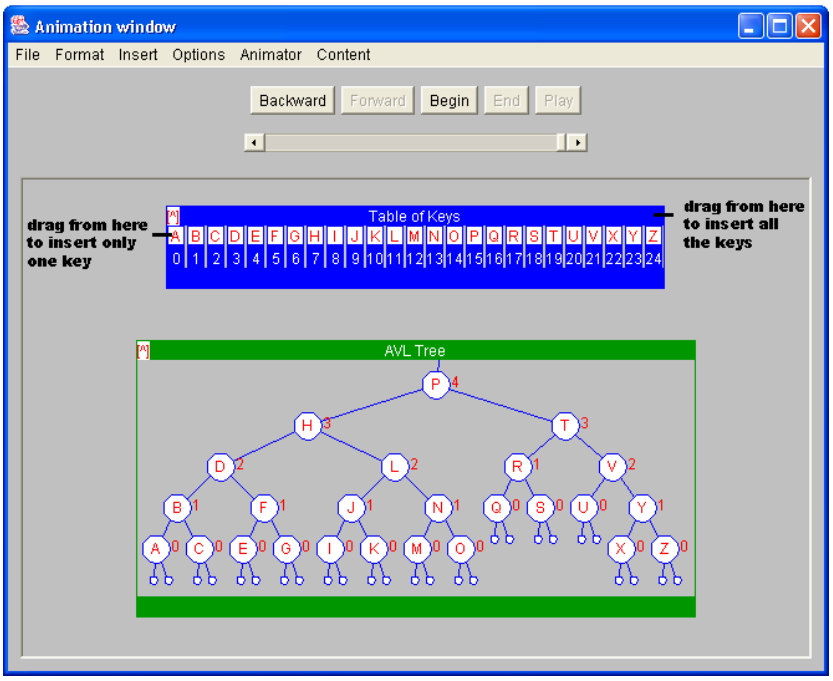

Figure 5: Keys can be inserted one at a time or the whole table at once

Matrix supports nested data structures of arbitrary complexity. It is possible to have complex data structures inside another one. For example, one can store arrays inside graph nodes, or trees inside array positions. As an example, see the B-tree implementation supplied with Matrix, in which arrays are nested inside tree nodes to hold the keys.

Some Fundamental Data Types, such as arrays or binary trees, have no semantics for inserting keys "into the structure". For such structures, keys must be inserted in a specific position, node, etc.

5.3.2 Deleting Keys and Nodes from a Data Structure

Objects can be deleted by using the delete command in the pop-up menu of the desired object. What the delete command actually does depends on the data structure or structure component it was called on.

Using the delete command on a visualization of a data structure will always remove the whole structure. The visualization of the structure is removed from the current frame. When a whole structure is deleted, it is not possible to undelete it by going backward in the animation.

The effect of deleting a part of a structure depends on the corresponding data structure. Deleting a tree node removes the subtree rooted at the deleted node. On the other hand,

invoking the delete command on a graph node causes that node (and all references to or from it) to deleted.

Some components of structures, such as array indexes, cannot be deleted. Moreover, the effects of deleting a node from an ADT depend on the ADT in question.

Objects can also be deleted by dragging and dropping them in a trash. A trash can be created using the Insert menu.

When used on an item in a CDT, the two ways to delete an item described above both act as if the deletion were performed on the FDT on which the CDT is based.

A different way to delete an item from a CDT is to hold the Shift key down while dragging the item away from the CDT and dropping it somewhere else (such as an empty part of the animation window). This deletes the item from the CDT. If the item is dropped on another structure, it is inserted as normal. Note that the CDT's delete routine is used to perform the delete in this case; for example, shift-dragging an item from a stack will always cause the topmost item on the stack to be deleted regardless of which item was dragged.

5.3.3 Other Operations

This section of the manual describes miscellaneous commands that can be used to manipulate the Matrix data structures.

To copy a subtree, merely drag the root node of the subtree to the desired position. If you copy a subtree to a different position in the same tree, the subtrees appear in DFS order and revisited nodes are marked as duplicate trees and shown minimized. The copied tree points to the original tree, so changes in either of the visualizations affect both the original and the copy.

In Matrix, it is possible to add vertices and edges to a graph as well as delete vertices from a graph. Vertices can be added by dragging and dropping keys onto the graph. Edges can be added by invoking the InsertEdge command on a vertex and left-clicking the boundary of a target vertex (not the key of the vertex). Inserting an edge can also be done by clicking the source node with shift-key held down and after that clicking the target node. The system will create an edge between the vertices and update the visualization. Vertices can be deleted by simply invoking the delete command.

Nodes and references in graphs and trees can be moved to point to another node by dragging and dropping them on the new target node. Tree nodes that have no references are removed. Note that tree references must be explicitly updated after a drag and drop operation using the Update References command in the Options menu.

The range on indices to be on focus in a table can be selected. This can be done by first clicking on a starting index of a range and the selecting the ending index of the range.

5.4 Animation

Animation in Matrix is controlled by using the control buttons of the animation control panel. The animation can also be controlled by using the slide bar of the control panel

to move the animation to the desired position. By default, each animation window will have its own animator. If a visualization of a data structure is opened in a new window (see section on pop-up menu commands for details), the two windows will have the same animator.

In Matrix operations are grouped into animation steps that can contain other, smaller steps. The smallest possible steps (atomic steps) may not have any visible effect on the visualizations. Moreover, it is possible to move forwards and backwards one atomic step at a time by pressing the arrows located at the ends of the slide bar. This is also the easiest way to go "inside" an animation step, since the beginning and end of each non-atomic step is an atomic step. The animation control buttons work on non-atomic steps.

5.4.1 Control panel

Figure 6: Animation control panel allows the control over the visualization

The **Backward** button will undo one operation (one enclosed animation step). If the data structure is modified while there are undone operations, these operations can no longer be redone.

The **Forward** button will redo one operation.

The **Begin** button will undo all possible operations. The beginning of an animation can be reset by selecting **Set beginning here** from the Animator menu.

The **End** button will redo all possible operations. The end of an animation can be reset by selecting **Set end here** from the Animator menu.

The **Play** button will play a step-by-step animation from the current animation state to the last animation state.

5.5 Customizing Visualization

The visualization of different data structures can be changed using the menus and pop-up menu. These commands do not change the underlying data structure; they only change how it is visualized.

Depending on the data structure, there are several different layouts for the structure. In addition, the representation can be, for example, flipped or rotated.

Some layouts allow special customization. For example, the visualization of graph and tree edges can be changed from directed to undirected and vice versa. In addition, the visualization of empty tree leaf nodes can be turned on and off and so on.

See the next section for more details.

6 Menu Commands

This section describes the various commands found in menus.

6.1 File Menu

File menu contains, for example, commands to open an animation or a structure, save an animation or a structure, close the current window, export an animation to SVG, print the current view and quit. (see Fig. 7)

Figure 7: File menu

New Window (Ctrl+N). Open a new animation window.

Open (Ctrl+O). Open a new data structure. Java *.class* files, saved Matrix animations and ascii text files containing string representation of a data structure can be opened. Matrix knows how to visualize saved animations and parsed strings automatically, but

java classes must implement Matrix visualization interfaces in order to be visualized correctly.

Known problems: animations are saved as serialized Java objects. This causes problems if object's class has changed after it is saved. The saved animation can be loaded back if and only if the class remains untouched from one release to another.

The following three text file formats are currently supported:

1. **edge list** (default) — In this format the edges of the graph are listed with one node pair per line. One node pair mathes with one edge in the graph.
2. **adjacency-list** — In this format each line contains a node and adjacent nodes of that node. The node and its adjacent node define one edge in the graph.
3. **array** — In this format each line contains one key. The key which is in the first line is put into index 0 in the array, the key in the second line is put into index 1 in the array etc.

See Appendix A for the examples of these file formats. Moreover, there is also a description of an extended text file format.

Save As...

1. **Serialization** — Open a dialoque where the animation in the current active window can be saved.
2. **ASCII** — Open a dialoque where the structures in the current window can be saved into ASCII file. This saves the structures as an extended file format which is described in the Appendix A. This also saves some information about visualization of every structure. Note, that you might still lose some information about the data structures when saving them as ASCII.

Close (Ctrl+W). Close the current active animation window.

Clear Clear the current active animation window. All the structures will be removed and the animator will be cleared.

Export... Export the current view or animation. Formats currently supported are the following:

1. **LaTeX** — export the current view in LaTeX format.
2. **SVG** — export the animation in SVG format.
3. **SVG compressed** — same as SVG but compressed with GZIP

LaTeX export creates Texdraw representation of the current view. The package needed for Texdraw can be found, for example, at

http://www.ibiblio.org/pub/packages/TeX/graphics/texdraw/

or

ftp://ftp.tsp.ece.mcgill.ca/TSP/texdraw/.

Exported SVG animations can be viewed, for example, with Adobe SVG Viewer browser plug-in that can be obtained from

http://www.adobe.com/svg.

Page Setup... Open the page setup dialog for printers.

Print... Print the current window.

Print animation... Print an animation from the configurations currently in the Animator. Each step of the animation is printed on its own page.

About. Show version information.

Quit. (Ctrl+Q) Exit the program.

6.2 Format Menu

Format menu holds the submenus where the font and the font size can be changed. (see Fig. 8)

Figure 8: Format menu

Font. Change the font used by visualizations.

Size. Change the font size used by visualizations.

6.3 Insert Menu

Insert menu holds the data structures provided with the Matrix. Selecting a structure will create a new instance of the selected structure to be visualized on the animation window. (see Fig. 9)

The structures in the menu are divided to three categories: FDTs, ADTs and Utils.

FDT stands for Fundamental Data Type, and includes the basic structures like binary trees, arrays, linked lists and graphs.

ADT stands for Abstract Data Types. ADTs are more complex structures that have a pre-defined set of operations. The implementation of an operation depends on the ADT. Dictionaries, priority queues etc. are ADTs in Matrix framework.

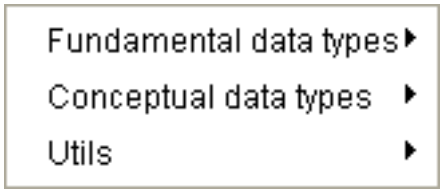

Figure 9: Insert menu

Utils are structures that are used to make the manipulation of ADTs and FDTs easier. The current Matrix utilities include a Trash, which can be used to delete objects.

6.4 Options Menu

The options menu contains special commands for simulation purposes. There is also debug tools.

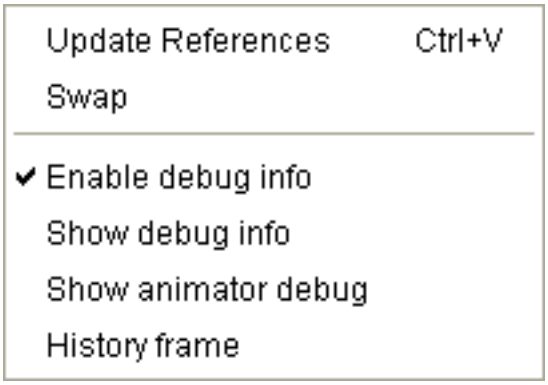

Figure 10: Options menu

Update references (shortcut Ctrl+V). Update references between objects and repaint the visualized data structures. One can change several references each at a time by just moving them to point into the desired target. The underlying structure (and thus, the visualization) is not updated until the update references operation is called.

Swap. Change the drag and drop operation semantics. There are two possible semantics: *insert* and *swap*. Insert is the default semantics and can be changed to swap with this command and vice versa. Insert semantics behaves as expected, thus moving an object from the source location into the destination does not change the original source structure (a := b;). While swap semantics is selected, dragging and dropping objects will cause the source and target objects to swap, *i.e.* change places (tmp := a; a := b; b := tmp;). Swap is only intended for keys and FDT data structures. ADTs can have unexpected behaviour with swap semantics.

Enable debug info. Switch debug output on or off.

Show debug info. Show debug information of all objects.

Show animator debug. Show debug information for animator.

History Frame Open the current visualization into the history frame that is a snapshot of the current state of the structure. The history frame does not change while animator operations are performed, thus the selected state can be compared with the visualization (*e.g.*, the previous state or the next state) in the active window.

6.5 Animator Menu

The Animator menu contains commands to control and modify the animator. Shortcuts are also available for the most used menu commands. See Section 5.4 for more details.

Figure 11: Animator menu

Backward (shortcut Ctrl + Arrow left) Undo one operation (one enclosed animation step). If the data structure is modified while there are undone operations, these operations can no longer be redone.

Forward (shortcut Ctrl + Arrow right) Redo one operation.

To beginning (shortcut Ctrl + Arrow up) Undo all possible operations.

To end (shortcut Ctrl + Arrow down) Redo all possible operations.

Play Play a step-by-step animation from the current animation state to the last animation state.

Set beginning here Set the current state to be the beginning of the animation. The previous states can no longer be reached.

Set end here Set the current state to be the end of the animation. The following states can no longer be reached.

6.6 Content Menu

At the moment this menu contains the prototypes of exercises implemented for the Data Types and Algorithms course (See TRAKLA2 application at Matrix home page for more details). Examples cover areas such as Basic Data Types, Tree Traversing, Dictionaries, and Priority Queues. In addition, an example case of code animation is included by introducing an exercise in which the user is asked to complete several tasks with Boyer-Moore-Horspool string matching algorithm.

This menu is going to disappear in the future releases. Instead, the content similar to this should be opened into the application from File->Open menu.

6.7 Pop-up Menu

The pop-up menu appears to be different for different items. In the following, the most general operations are described. Some other items may have additional operations.

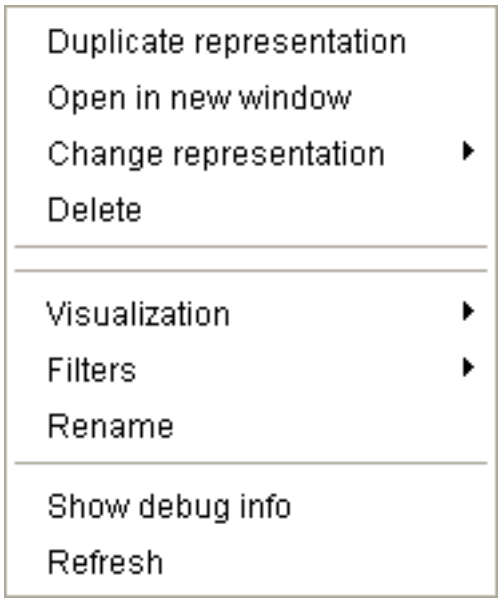

Figure 12: pop-up menu for a tree

Duplicate representation. Create a new visualization of the data structure in the current animation window. Changes in the new visualization affects the original and vice versa.

Open in new window. Open a new visualization of the data structure in a new animation window. Changes in the new visualization affects the original and vice versa.

Delete. Invoke the delete method for this object. By default this removes the selected structure or component from the underlying data structure.

Change representations. Change the layout for the data structure.

Visualization submenu contains commands that directly modify how the data structure is visualized. See the section about this submenu for further information.

Filters submenu contains additional methods that, depending on the data structure, can be used to filter out the structure's details, or select only a part of it to be represented. See the section about this submenu for futher information.

Rename. Rename a data structure. This has effect only on keys, data structures with a header, or labeled nodes. This command is also applied to modify the value of a key.

Rename all keys (Tables only). Rename all keys of the table. This opens a dialogue where the new keys can be entered. The keys must be separated by space character.

Labeled (Nodes only). Enable or disable the label besides the node.

InsertEdge (Graph vertices only). Insert an edge between two vertices (after the destination vertex has been clicked). (Inserting an edge can also be done by clicking the source node with shift-key held down and after that clicking the target node.)

Show debug info. Dump debug information for the selected data structure.

Refresh. Refresh the visualization. As a side effect, this will create new random keys for a table of random keys.

Call. This menu appears only if user-defined methods are defined for this object. Methods without parameters within this drop down list can be invoked by selecting the desired method.

6.7.1 Visualization Menu

Figure 13: Visualization menu

Minimized. Minimize or maximize a visualization.

Alive. Enable and disable visualization's response to simulation operations. If a visualization is not alive, it does not respond to algorithm simulation operations (for example dragging and dropping).

Enable. Enable and disable direct access to the subcomponents of a visualization. If a visualization is not enabled, its subcomponents cannot be accessed through GUI.

Titled. Turn data structure title bar on or off (see Fig. 14). To get title bar back for an array, you have to open the pop-up menu by clicking the right mouse button near the bottom edge of an array. For other structures, you can just click the right mouse button above an empty place in the structure.

Figure 14: The array in Fig. 3 without the title bar

Rotated. Rotate the visualization (see Fig. 15).

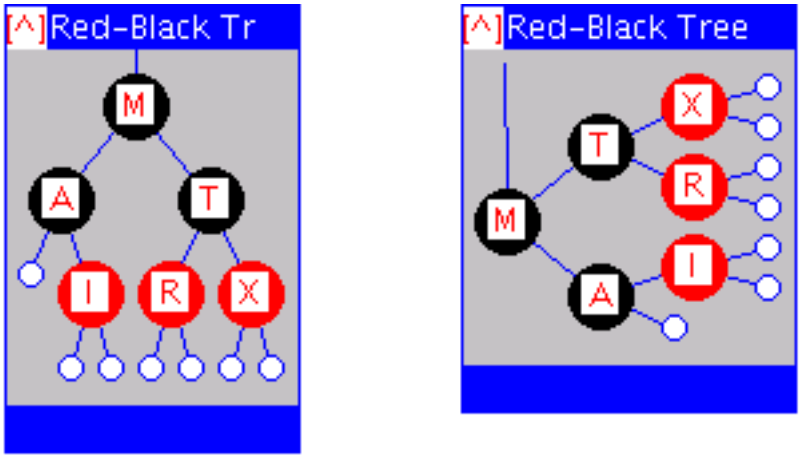

Figure 15: On the left is the original visualization and on the right is the corresponding rotated visualization

FlipX. Flips the X coordinates of the visualization (see Fig. 16).

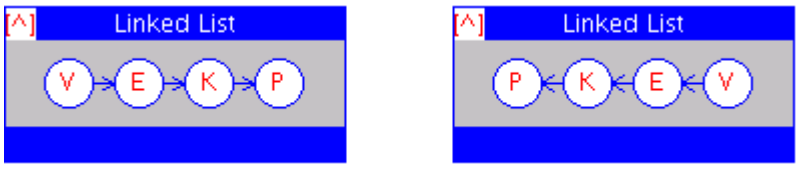

Figure 16: Original visualization on the left and x-flipped on the right

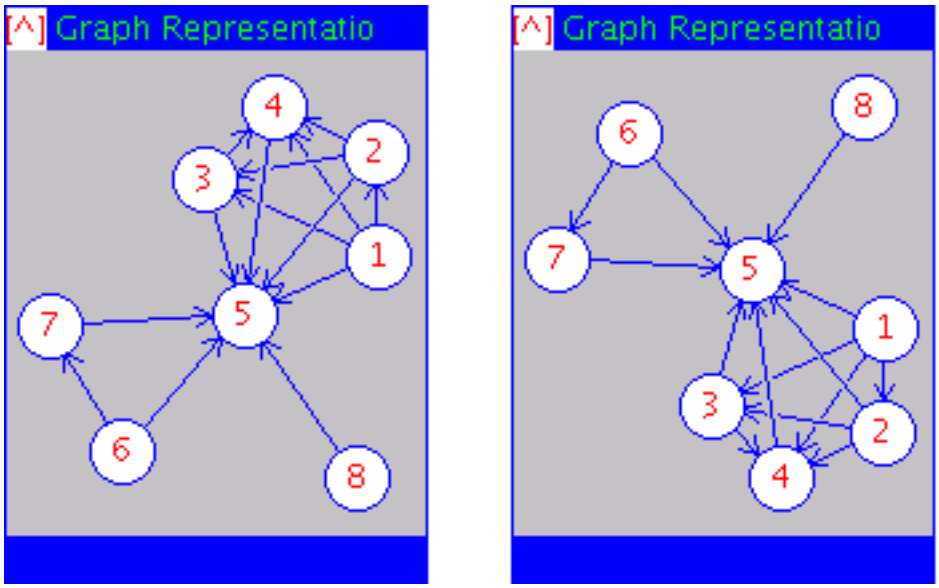

Figure 17: Original visualization on the left and y-flipped on the right

FlipY. Flips the Y coordinates of the visualization (see Fig. 17).

Indexed (Arrays only). Turn array indexes on or off.

6.7.2 Filters Menu

The contents of the filters menu depend heavily on the data structure. Practically every structure has a different filters menu. Figure 18 shows filters menu for a tree and Figure 19 for an array.

Figure 18: Filters menu for a tree

Directed (Trees and Graphs only). Select edges appear as directed or undirected.

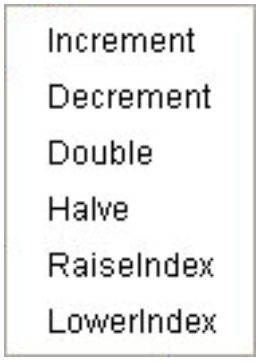

Figure 19: Filters menu for an array

EmptyLeaves (Trees only). Show or hide empty leaves.

DFSvalidate (Graphs only). Validate the graph in DFS order. Otherwise, validate in BFS order.

BackEdges. Show or hide back edges for graphs.

ForwardEdges. Show or hide forward edges for graphs.

CrossEdges. Show or hide cross edges for graphs.

Increment (Arrays only). Increase the size of an (dynamic) array by one.

Decrement (Arrays only). Decrease the size of an array by one.

Double (Arrays only). Double the size of an array.

Halve (Arrays only). Halve the size of an array.

RaiseIndex (Arrays only). Shift array indexes right by one.

LowerIndex (Arrays only). Shift array indexes left by one.

7 Adding New Data Structures

The programmer's manual, located in $MATRIX/docs/Manual/Programmers/ describes how to add new data structures, create new visual concepts and modify your own data structures to support Matrix animation features.

You can use the Open command in the File menu to open your own data structures in Matrix.

8 FAQ

Question: What is a "static binary tree(8)"? It actually has nine nodes.

Answer: It is an FDT that corresponds to the binary heap implementation. There is no heap operations, but the underlying structure is the same. One can select "new representation" and change it to "array" from the pop-up menu. Now the structure should be viewed as (1) a binary tree and (2) an array. By changing the size of the array (use the "->" hot spot at the title bar) should also change the number of nodes in the binary tree. Similarly any modification to the binary tree (e.g. insertion of new key) should also change the array representation.

Question: When inserting into an FDT list, the element is added into the front of the list, regardless of where in the background you drag it. Can I add into the end of the list?

Answer: This answer may vary between different implementations, but what comes to the prototype implementation the answer is yes. If a *key* is dragged into a *node* within the *list*, then the key will be inserted after that node in the list. If it is dragged in the background, it will be added into the front of the list. In general, the title and background here represent the container that corresponds to the "whole structure". Here, the whole structure list has an insert method that is implemented in such a manner that the new key is put at the front of the list. On the other hand, each node can act as a component and implement an insert operation of their own. Thus, insert into the node will add a new node after the corresponding node.

Question: I'm using Matrix in Windows but it seems that the pop-up menu doesn't work.

Answer: We have noticed that there are problems with the pop-up menu in Windows. Here are two solutions that might work.

Solution 1:

1. Click the right mouse button to bring out the pop-up menu of the current element.
2. Then click the left and the right mouse buttons at the same time on a desired choice. This should bring out again the pop-up menu.
3. Now click the left mouse button on a desired choice.

Solution 2:

1. Press the right mouse button on a desired element and keep it pressed.
2. Move the mouse cursor little bit while still pressing the right mouse button.
3. Now release the right mouse button. This should bring out the pop-up menu.
4. Now click the left mouse button on a desired choice.

A APPENDIX - Text file formats

As mentioned in the File menu -section the following three text file formats are currently supported:

1. **edge list** (default) — In this format the edges of the graph are listed with one node pair per line. One node pair mathes with one edge in the graph.
2. **adjacency-list** — In this format each line contains a node and adjacent nodes of that node. The node and its adjacent node define one edge in the graph.
3. **array** — In this format each line contains one key. The key which is in the first line is put into index 0 in the array, the key in the second line is put into index 1 in the array etc.

The text files should contain the following headings and notations, respectively.

```
#matrix graph
1 2
1 3
1 4
1 5
2 3
2 4
2 5
3 4
3 5
4 5
```

```
#matrix graph adjacency-list
A:B C D
B:C
C:E
D:E
E:B
F:G
G:H
H:F
```

```
#matrix array
A
B
C
D
E
F
```

These examples are located in the $MATRIX/code/examples/ directory.

There is also a new text file format which makes it possible that one file can contain arbitrary number of structures. It also enables that the keys in the structures can now contain, for example, space characters. Moreover, with new format, the main structures can have inner structures and furthermore these inner structures can have their own inner structures etc.

The following example is also in the $MATRIX/code/examples/ directory.

Example:

```
#matrix structures                //heading of the file
test#1                            //name of the first main structure
#matrix graph adjacency-list      //type of the structure
a:test#1_1 c                      //keys in adjacency-list format
test#1_1:e                        //test#1_1 is an inner structure
c:d
e:test#1_2
#EOS                              //end of structure -character
  test#1_1                        //description of the inner structure
  #matrix array
  a
  b
  c
  #EOS
  test#1_2                        //another inner structure
  #matrix graph adjacency-list
  key1:key2 key3 key4
  key2:test#1_2_1
  key4:key6 key7
  #EOS
    test#1_2_1
    #matrix array
    d
    e
    f
    #EOS
test#2                            //another main structure
#matrix array                     //type of the structure
asdf                              //keys in array format
test#2_1
qwerty
#EOS
  test#2_1                        //inner structure
  #matrix array
  aa
  bb
  cc
  dd
  #EOS
```

Names of the main structures must end with “#number” (e.g.structure#1) because they are separated from inner structures with this ending. Names of the inner structures should be chosen so that it is easy to recognize their parent structures (e.g. structure#1_1). The Description of the structure comes after the name of the structure.. This can be either in adjacency-list format or in array format because these formats are supported in this extended file format. In the end of the description comes “#EOS” which marks the end of that structure.

The keys can be almost anything. If the key contains characters “ “ (space character), “_” or “#” they must be marked with special character so that they are interpreted correctly when opening the file. Special character is “ ’ “ and it must be inserted right before the wanted character (e.g. inner’ structure, a’_b, ’#matrix and I”m). When opening the file these special characters are removed from the keys and the original keys are obtained.

It is also possible that the graph description can contain dublicates. If many nodes have the same key in graph, these dublicates can be marked by adding “_number” in the end of them. First occurrence of the key is marked as “key”, second one is marked as “key_2” etc. During the opening of the file these suffixes are removed and the original keys are obtained and added to the graph.

To make the file more readable the lines can be indented with space characters. With help of this it is possible to indent inner structures more than outer structures and this way to represent the hierarchy of the structures.

Moreover, lines can contain additional comments, which are skipped when loading the file. Comments must start with string “//”. Space characters are also removed from the end of each line. Empty characters before the comment must be space characters.

Also some information about visualization of each structure (representation, rotated and minimized) can be saved into file but this is optional.

Example:

```
...
test#1                              //name of the structure
#representation layered graph
#rotated true
#minimized false
#matrix graph adjacency-list        //type of the structure
...
```