CALIFORNIA STATE UNIVERSITY, SAN BERNARDINO

School of Computer Science
And Engineering

CSE 455

{AlgorithmA }; 2011

How to Use the Authoring System

CSE 455, Inc.
CEO: Dr. Concepcion
Project Manager: Anthony Thomas
Assistant Manager: Erwin Soekianto
Team Leads: Matthew Compean and Patrick Ridge

Table Of Contents

1 About the Authoring System

The purpose of the Algorithma authoring system is to help users to get a feel for how C++ code works. The authoring system does this by allowing users to edit existing algorithms, provided by the authoring system in a drop down menu below the code, or create their own algorithms and then the authoring system will give a visual demo of the code. Either C++ or pseudo code can be entered into the system, but note that the pseudo code follows the standards which can be found in Fundamentals of Algorithmics, the book used for CSE 431. The authoring system currently has three different methods for animating code.

1.1 Simple Sorts

The simple sorts animation class gives the user a single array to manipulate. Single instances in the array can be manipulated and moved around within the array based on various sorting techniques. Note that simple sorts can not make multiple arrays.

1.2 Array-Intensive Sorts

Array-Intensive Sorts functions similarly to simple sorts, but it also provides the user with the ability to break down and merge arrays. The animation will add a line between blocks to show a split into two or more arrays and then remove lines to show that arrays have merged.

1.3 Basic Constructs

Basic Constructs is different from the other two animation classes. Instead of providing the user with an array, Basic Constructs will display variables for the user, allowing them to manipulate and track variables as the code executes.

2 Adding to the Authoring System

The authoring system consists of animation classes or modules which can be created and added to the system.

2.1 What is a Module?

A module is a particular animation class built into the authoring system. When we were first

'hired' to create this system, all we were told was "I want to be able to enter pseudo-code and have it animated". Well, we knew this to be impossible. JsAnim, while an interesting library to be sure, simply does not have the dynamic capability to do that of its own accord. Heck, JsAnim is fully absolute location based, not even relative - meaning that having the animations be different depending on the code applied was going to be tough even for one type of animation. With this in mind, we decided to create modules - animation classes that a user has to select before inputting code. The idea broke the authoring system down into manageable chunks.

From 2010, you will be receiving 3 modules - Simple sorts, Complex (array intensive) sorts, and basic constructs. Simple sorts will be fairly well tested and 'done'. Basic Constructs was thrown on us at the last minute, but since it's so simple it should work fairly stably. Complex (array-intensive) sorts caused some problems, but in the last week or so we came up with an animation 'class' that uses 1 dimension of movement (instead of 2 like the regular merge sort) that worked decently.

Additionally, you will see two master files, pseudoToJs.js and cppToJs.js. These are the beginnings of magic - though they have a long way to go. While these will do ok transforming basic c++ and pseudo-code (side-note, we used the pseudo-code from Algorithm Analysis CSE431) to javascript, they do stumble on certain cases (a few examples - opening curly braces must be on their own line, array support is a bit 'simple', etc). It will be your job to build on these.

2.2 Creating a Module Part I

Modules consist of several files and edits. You'll need graphics for your module (or steal from other algorithms) and these will go in the images directory. You'll need a php file for your module in the pages directory (ex: author_basic.php). You'll need a JS file in the js/algorithm directory (ex: authorSimpleSrc.js). You'll need a CSS file in the css/algorithm directory (ex: authorSimpleStyle.css).

Now open up index.php and add your php filename to the whitelist ($validauthor = array('simplesort', 'complexsort', 'basic');).

Open up header.php and add your new pages to the end of that massive switch/if else statement (ew). (Include the pseudoToJs and cppToJs files if you desire - we are attempting to use them in all modules, but certain modules may require specific 'different' coding).

```
EX:     else if($_GET['type']=='basic')
        {
    print <<<EOF
        <link rel="stylesheet" type="text/css" href="css/algorithm/authorBasicStyle.css" />
        <script type="text/javascript" src="js/algorithm/authorBasicSrc.js"></script>
        <script type="text/javascript" src="js/pseudoToJs.js"></script>
        <script type="text/javascript" src="js/cppToJs.js"></script>
        <link rel="stylesheet" type="text/css" href="css/algorithmLayout.css" />
EOF;
        }
```

Finally, in the bottom of header.php, add a link:

EX:

```
<li><a href="#">Authoring System</a>
 <ul>
 <li><a href="index.php?page=author&type=simplesort">Simple Sorts</a></li>
 <li><a href="index.php?page=author&type=basic">Basic Constructs</a></li>
 <li><a href="index.php?page=author&type=complexsort">Array-Intensive
   Sorts</a></li>
```

2.3 Creating a Module Part 2

Ok, now your files are installed. But wait, what files? I haven't covered how to logically create a new module! Here's how I did it:

I would take a currently existing algorithm, and make it dynamic (e.g. I took insertion sort to make the simple sort module, I took the merge sort to make the complex/array intensive module). You need to make this algorithm actually run javascript code - so create that particular algorithm in javascript. Attempt to integrate the existing animation functions within your javascript version of the algorithm. Don't give up here - it can be tough... you may have to alter some of the animations or create your own.

Once you have the algorithm running through javascript while inserting animations within that code you're ready to continue. Now you need to create pseudo-code and c++ for your algorithm, running it through the converters to turn it into javascript. Fix / make changes to the converters as needed to convert your code properly. Now you need to make a function that takes the javascript code from the converters as input and adds your animation code to the javascript. (look in the other modules for examples).

If you are having issues getting the animations to run in sequence with delays, use the timer system from the array intensive module (which I believe we'll also be using in the basic constructs). It's pretty simple - it adds the animations to an naMoves array, and then a timer is created running every 2 seconds (or something like that) that takes the first element in the naMoves array and executes it using eval (so you'll be pushing actual javascript code as a string onto the naMoves array).

And that's really it.