Binary Search Tree Visualization Using Scalable Vector Graphics and HTML

By: Isaac O'Beirn
For: SEG 2900
Submitted to: Prof. Turcotte and Prof. Some
Date: December 20th, 2017

Introduction

This report describes the steps undertaken to generate a Binary Search Tree (BST) using Scalable Vector Graphics (SVG) in an HTML document. A random tree was to be generated and printed to the user. Given a contextual question about the tree, the user would answer by clicking on a node. If the correct node was selected, it would turn green; if not, it would turn red and all possible correct answers would turn green. This was to prove the concept that SVG elements could be used for displaying such objects and allow for better interactivity. These SVG elements could be resized to fit any window and certain tags could be added for additional accessibility.

The discussed code can be found on GitHub by following the reference [3].

The Task

A series of questions are selectable by the user, such as: “Find the tree Root”. At any time, the user could change the question or rebuild the random tree. The nodes had to change colour when clicked on, depending on the question and what the correct answers are.

The tree must be printed to the webpage nicely whilst maintaining the BST structure. That is to say, left children must be to the left of the parent, etc. It was chosen to print the tree vertically, where the root is at the top middle of the figure as shown bellow.

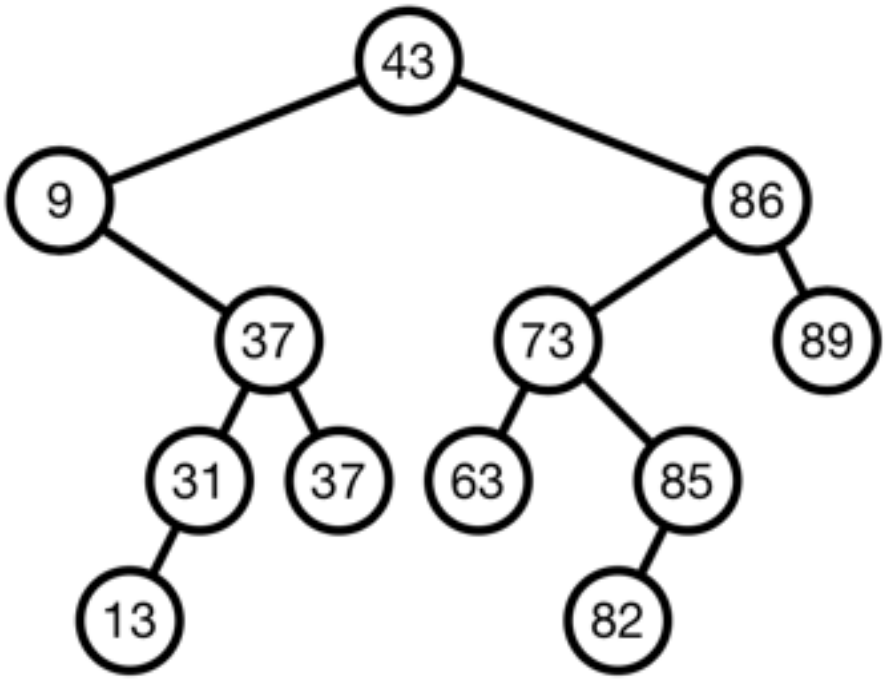

Figure 1 - Random BST where “43” is the root.

The Result

From Figure 1, it can be seen that the Left child of the Root (“43”) and Right child are placed on either side. Each subtree is spaced as to not overlap other nodes, all the while conserving space as much as possible. There is a minimum space between each node, as well as between each row (level, or depth) of nodes.
The following figures demonstrate the results of selecting the proper and the incorrect answer to the question “Find the Root”, respectively.

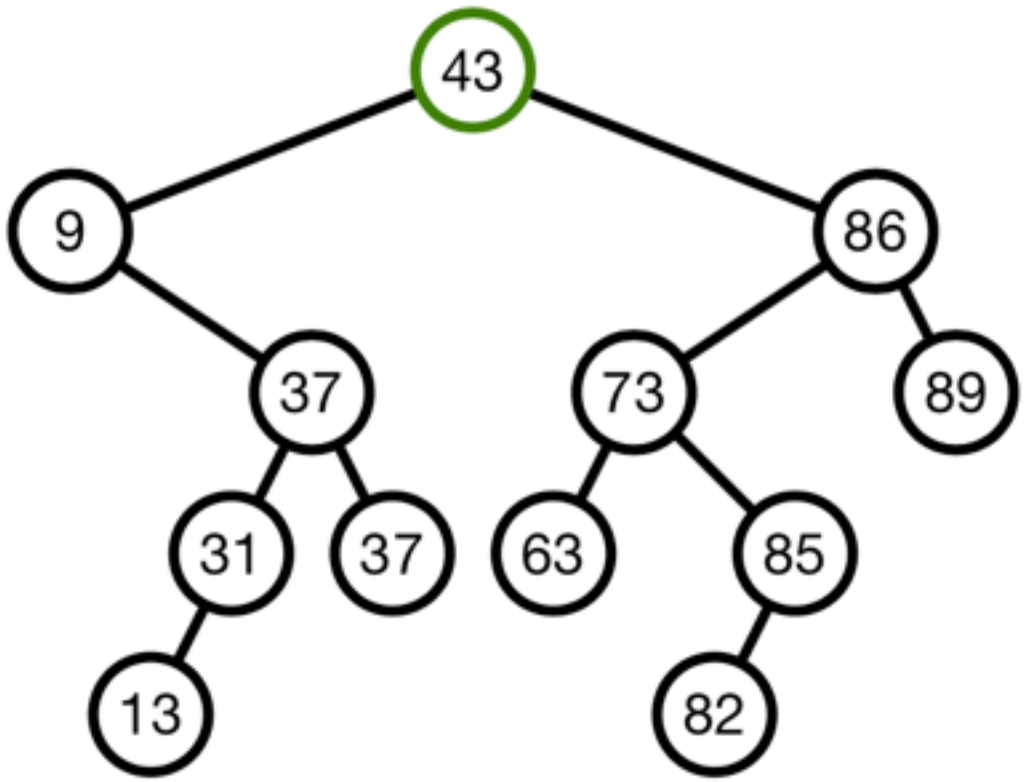

Figure 2 - Node “43” was selected to answer the question “Find the Root”.

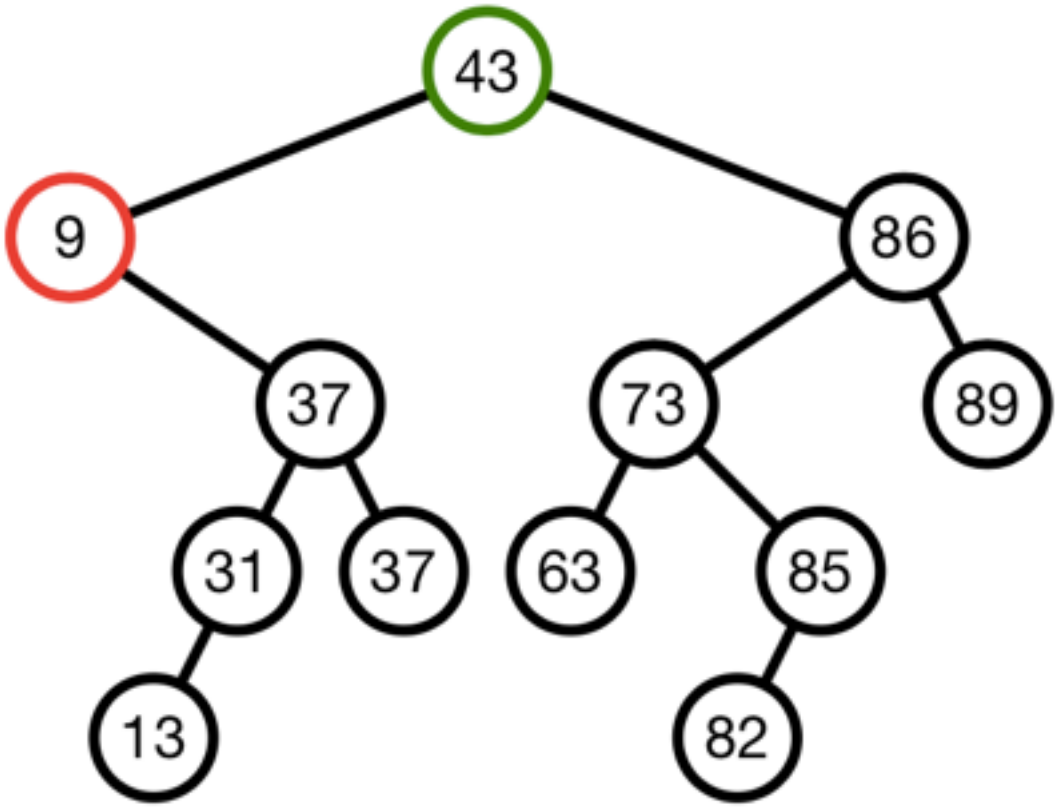

Figure 3 - Node “9” was selected to answer the question “Find the Root”.

Notice the node “9” is red in Figure 3 because it is not the correct answer. Node “43” is green to show the user what the correct answer was.

The Code

Standard SVG libraries were used to display these figures. A template HTML page is first loaded, then JavaScript (JS) functions generate the tree data structure, populate it, and print it to the user. The drawing function injects HTML tags to the template HTML page. These tags are SVG elements such as circles and lines. Lines must be added first, then circles, then the text since SVG lacks the ability to set drawing layers.

Basic CSS attributes were set in the CSS file for quick modification of element groups. Here, the colour for correct answers, wrong answers, etc. can be changed.

A Floating Coordinate algorithm [1] was used to calculate the appropriate node X and Y positions in the figure. This algorithm completes in O(n) time and serves to efficiently space out the nodes per row.

“draw_Random_Tree” first clears the current figure in the HTML page, then generates the tree.

“random_BST” is used to generate the random BST. The number of nodes is constant and can be set by modifying the template HTML page. The numbers are random integers from 1 to 100, inclusively.

Once the tree is created, “draw_Random_Tree” calls “float_Coord” to assign X and Y positions to each node. These positions are stored in the node elements themselves.

Then, the node SVG elements are created and injected into the HTML through a call to “draw_Nodes”. Finally, the SVG view box is created according to the current tree dimensions. It is sized such that the tree is in the centre. Margins are added to the figure.

If no right answer is found in the tree to the question at hand, the tree is regenerated and redrawn. This must be checked post- drawing since the correct answers are stored in the SVG tags as classes. Each node contains a list of classes which it belongs to, such as “root”, “left_sibbling”, etc. New classes can be added by modifying the “buildClasses” function.

Changing the question will not reprint the tree, but it will reset the colours. New questions can be added by modifying the “Questions” object, which is treated as a dictionary where the key is the correct answer and the value is the question string.

Clicking on the “New Tree” button in the HTML page will generate a new tree without changing the current question.

Discussion

Currently, it is recommended to not exceed 25 nodes per tree for risk of creating figures far too large. Larger trees are “zoomed out” to display entirely in the window, thus a larger tree may result in nodes too small to read.

Since the nodes contain a list of classes, which includes the correct answer, users can view the source code and find the correct answers should they choose. Future improvements may include kidding these attributes in each nodes JS object instead.

Accessibility can be added to these elements by encapsulating each node group (Circle, Text, and Line element) in a “graphics_symbol” and building an Accessibility Tree listing each of these nodes and their values. This would allow certain Accessibility APIs to navigate the figure [2].

References

[1] V. Borovskiy, J. Müller, M.-P. Schapranow, and A. Zeier, “BINARY SEARCH TREE VISUALIZATION ALGORITHM.”

[2] “SVG Accessibility API Mappings.” [Online]. Available: https://www.w3.org/TR/svg-aam-1.0/#include_elements. [Accessed: 20-Dec-2017].

[3] I. O’Beirn, “SVG-BST,” GitHub, 2017. [Online]. Available: https://github.com/eyezaak011/SVG-BST. [Accessed: 20-Dec-2017].