# Visualizing Trees

Nchinda Nchinda and Ray Hua Wu

December 13, 2017

**Abstract**

We implement an interactive tool to aid in the understanding of binary search trees, their geometric views, and x-Fast Trees. Through commanding operations on the interface and watching the animated output, a user can visually learn about the components and methods of the data structures.

## 1 Introduction

### 1.1 Structures

We implement two types of trees in our visualization. Both of these trees aid in optimizing the ordered-dictionary operations: insert, delete, search, predecessor, and successor. (The problem of optimizing for these operations is often known as the Predecessor Problem.) If only insert, delete, and search were desired, hashing is a great solution. But predecessor and successor are expensive operations on a hash set ($O(n)$). Trees help bring the runtimes of successor and predecessor operations to $o(n)$.

#### 1.1.1 Binary Search Trees

Binary search trees allow insert, search, delete, successor, and predecessor operations to occur in time on the order of the height of the tree. The height of a tree could be as large as its number of nodes, so without further restrictions this is not an improvement over a more naive structure, but if the binary search tree is balanced, and thus the height is logarithmic in the number of elements in the tree, these operations have runtime $O(\log n)$ for $n$ elements. [1]

#### 1.1.2 Geometric Views of Binary Search Trees

Certain properties of binary search trees can be well expressed in a geometric representation. In this representation, points are plotted in a grid with an x-coordinate representing the key of a node and a y-coordinate representing its access order. [2]

One could then insert a point set to this geometric representation such that the set of points on a row is precisely the elements visited in a search query for the element represented on that row. A set of such points represents a valid BST if and only if the set of points is an Arborally Satisfied Set, that is, a set of points in which any two points not in the same horizontal or vertical line have another point on the lattice rectangle spanned by the points.

The Greedy Algorithm to create this Arborally Satisfied Set is to consider the point set one row at a time, from bottom upwards, and add points to the row on each side, recursively, to satisfy pairs of points that are not arborally satisfied yet. [3]

#### 1.1.3 vEB Trees and x-Fast Trees

If we restrict the keys that we insert into the structure to integers, though, we can often do better than a Binary Search Tree, for instance, using a van Emde Boas Tree (vEB Tree). Such a tree specifically works best with a data set of integers not too spread out. In particular, the van Emde Boas Tree is unusual among structures in that rather than having a standard amount of memory dedicated to each piece of stored data, pieces of memory are assigned to each possible key in the universe and keys that have been inserted and not deleted are highlighted among these pieces of memory instead. Hence, operations on van Emde Boas trees are really measured with respect to the size of the universe, $U$. In particular, all of insert, delete, search, successor, and predecessor are $O(\log\log U)$-time operations.

One of the key aspects of the van Emde Boas Tree is that trees store their minimum and maximum non-recursively. Hence, minima and maxima can be obtained directly, not necessitating a recursive call. This is important for keeping ordered-dictionary operations $O(\log\log U)$ rather than $O(\log U)$, since thus only either a summary or cluster recursion is necessary, and not both. [4]

One major disadvantage of the vEB Tree is its enormous space usage, particularly when the data set has a high range. The x-Fast Tree is a step towards alleviating this space issue, trading this off for runtime of certain operations. It is a step on the way to the y-Fast Tree, a strictly more efficient data structure than the vEB Tree which builds on the x-Fast Tree concept. In the x-Fast Tree, 0s (non-inclusions of elements) are not stored, and root-to-1 paths are saved separately. Hence, even though the space usage would be similar to that of the van Emde Boas tree if many elements were stored, a small number of elements with a large range will not take memory to store many 0s representing unincluded elements. That is, space usage is now $O(n \log U)$ for storing $n$ elements out of a universe of size $U$. Searching an element takes $O(\log\log U)$, like in a vEB Tree, but inserting an element is now $O(\log U)$, slower than in a vEB Tree. [5]

## 1.2 Javascript Tools

This visualization project utilizes React.js and d3.js.

### 1.2.1 React.js

React.js is a JavaScript User Interface (UI) framework, maintained primarily by Facebook. React.js operates on a rendering system. Here is some example usage of React.js. Note that this is the Javascript file, specifying HTML.

```
render () {
  return (
    <main id="xfast">
      <h1 id="title"><a href="https://en.wikipedia.org/wiki/X-fast_trie">X-Fast
      (vEB) Tree View</a></h1>
      <form onSubmit={this.insertElement}>
        <p>Insert an element</p>
        <div className="tooltip">?
          <span className="tooltiptext">An X-fast tree improves upon a vEB tree
          by not storing elements that don't exist.</span>
        </div>
        <input value={this.state.newElement} onChange={this.changeElement}>
        </input>
        <button type="submit">Insert</button>
        <span>{Number(this.state.newElement).toString(2)}</span>
      </form>
      <div id="bittricks">
        <button onClick={this.halveBits}>Halve bits</button>
        <button onClick={this.doubleBits}>Double bits</button>
        <span> Current Size: {this.state.root.bits} bits </span>
        <span> Max Value: {Math.pow(2, this.state.root.bits) - 1} </span>
      </div>
      <svg id="veb" className="graph">
        <g id="links"/>
        <g id="nodes"/>
        <g id="values"/>
      </svg>
    </main>
  );
}
```

In the binary search tree visualization, an additional feature called Redux is used to link React interfaces together, namely the standard and geometric views of binary search trees.

#### 1.2.2 d3.js

d3.js is a JavaScript library for producing interactive data visualizations. It is, in a sense, a wrapper around SVG. It operates via binding data to visual elements.

Whereas SVG is made to be static, d3 allows a chain of alterations of the attributes of a graphical element, updated on the spot. An example from StandardBSTGraph.js is

```
node.enter()
    .append('svg')
    .attr('class', 'node')
    .attr('x', (d) => d.x)
    .attr('y', (d) => d.y)
    .attr('width', '40')
    .attr('height', '40')
    .attr('opacity', 0);
```

, which defines action for new data, in which a circle is later drawn (using similar attributes). State change procedures could be specified on top of these elements, for instance

```
standard.selectAll('svg.node')
        .transition()
        .duration(500)
        .attr('opacity', 1)
        .attr('x', (d) => d.x)
        .attr('y', (d) => d.y);
```

, later in StandardBSTGraph.js. d3.js also allows for HTML query selection mechanisms to process changes.

```
let node = standard.select('#nodes').selectAll('svg.node')
                .data(nodes, d => d.id);
```

### 1.3 Other Tools

This project makes use of Sass, or Syntactically Awesome Style Sheets. Sass allows more typical elements of code, like variables and operators, in the generation of CSS.

## 2 Repository Structure

The essential parts of the main folder of the tree visualizer repository are `app.js`, `index.html`, the `public` folder, and the `client` folder.

### 2.1 app.js

The script `app.js` is the starting point called by `node.js` to begin running the app. Other Javascript resources are dispatched from here.

### 2.2 index.html

As with any online resource, the home page lives in `index.html`. Here, though, the file is bare-bones, as it will be filled in later with our Javascript components, via the scripts called in `index.html`.

### 2.3 public folder

This folder contains an overall stylesheet as well as `main.js` (in the `js` folder), which coordinates most visualization boilerplate.

### 2.4 client folder

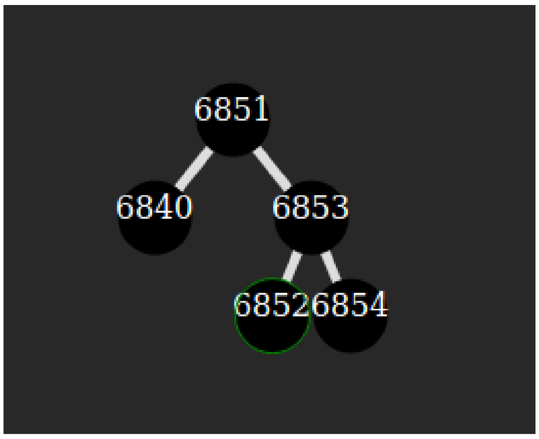
Most code immediately relevant to visualizations resides in the `client` folder, and in the `js` subfolder. (The `sass` folder contains style sheets.)

The files `StandardBSTNode.js`, `GeometricBST.js`, and `VEBNode.js` contain implementations of data structures, and `StandardBSTGraph.js`, `GeometricBSTGraph.js`, and `XFastGraph.js` contain procedures for visualizing the structures built in the implementations. `TreeView.js` is used to convert vEB Trees to x-Fast Trees for display.

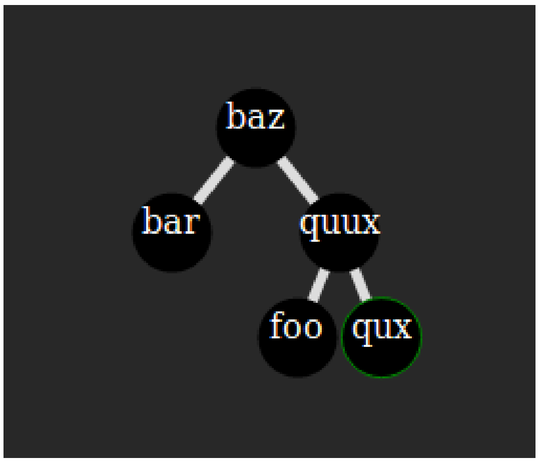
## 3 Features of the Interface

### 3.1 Binary Search Tree

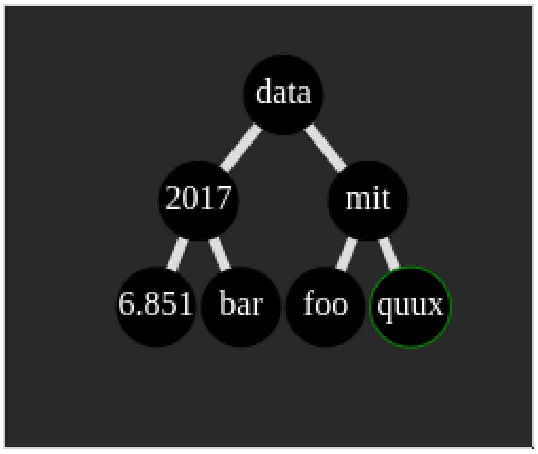
The interface allows the user to insert items into a binary search tree, in a text box near the top in the "Standard View" pane. These items can be consistently numbers, consistently strings, or even both. If numbers are inserted, the tree will assume numerical comparison as the comparator for building the tree.



If strings are inserted, the tree will assume lexicographical comparison as the comparator.



If both numbers and strings are inserted, the tree will sort the numbers using numerical comparison, sort the strings using lexicographical comparison, and consider all strings to be greater than all numbers.



The insertion happens upon clicking "Insert", and is animated. The animation will usually just be the placing of a node in a leaf position in a tree, but will sometimes show a large quantity of nodes changing position, specifically when the tree needs to be rebalanced.

As the binary search tree is built, the geometric view of the tree is updated in sync to inserts on the standard pane, via processes described in the Introduction. This geometric view comes with a Greedy Algorithm visualization, which is automatically performed upon insertions.

Two sequences are pre-defined for one-click availability: an increasing integer sequence and a bit reversal sequence. The insertions of these sequences' elements are animated one after the other when applying these pre-definitions.

If the size of a display is not optimal, the user can click and drag the visualization to pan across it, or scroll to adjust its zoom level.

## 3.2 x-Fast Trees

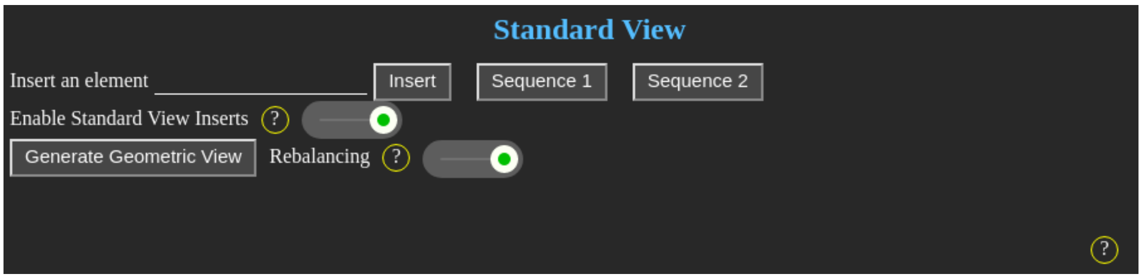
The x-Fast Tree tab allows one to visualize insertions in an x-Fast Tree.

Upon insert of an element, the path for that element in the tree is depicted. One could double or halve the number of bits the tree represents to obtain the desired tree size, although at as low as 16 bits, computational resource usage of a full visualization becomes concerning enough that our visualization prohibits it. (Even if that could be drawn, the resultant tree would be too unwieldy anyway.)

As with the binary search tree visualization, the user can click and drag the visualization to pan across it, or scroll to adjust its zoom level. This is particularly crucial here for the 8-bit visualization.

## 3.3 Tooltips

Each question mark (?) on the interface indicates a location where a tooltip can be viewed upon mouse hover. These provide at-the-moment help to users wondering about unfamiliar features.



# 4 Challenges

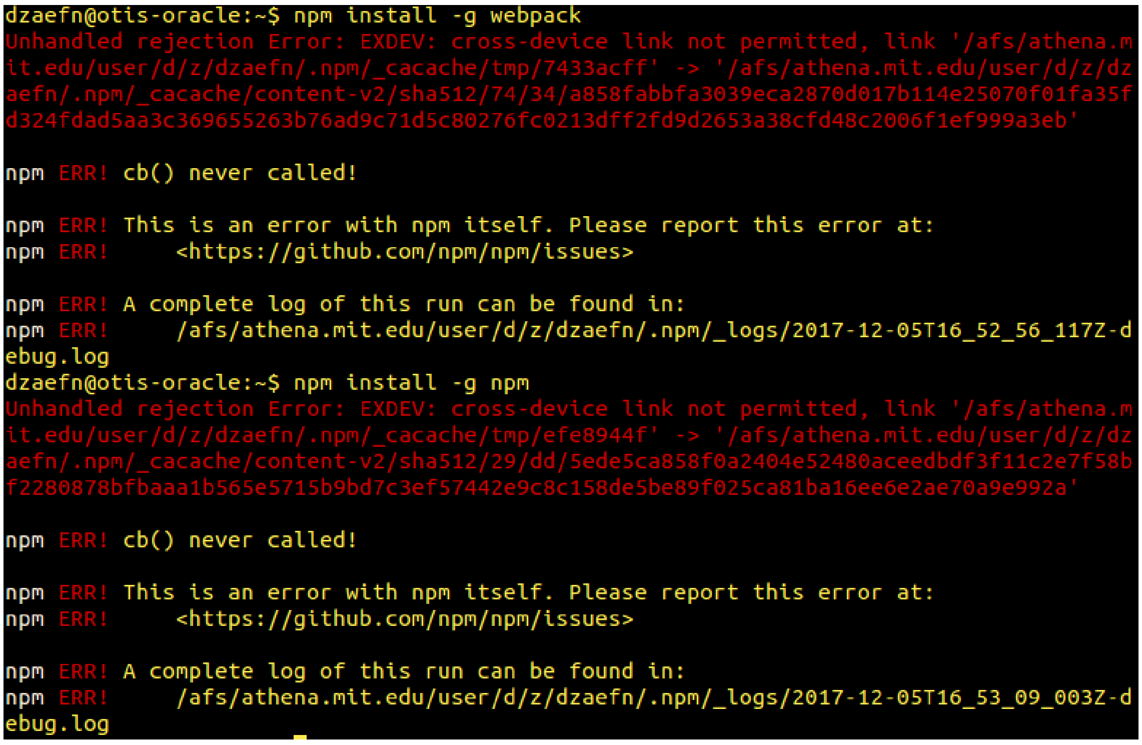
## 4.1 Complexity of Dependencies

Package complexity was a major challenge in this project. This project incorporated several different packages, which required not only gaining familiarity with these packages, but also understanding their interactions with each other. The process of debugging when using many packages was often tedious, as there are often numerous points of failure to check.

#### 4.1.1 Athena is Incompatible with npm

(For some context: early in this semester, one of us (Ray Hua) had the misfortune of having a functioning laptop enter a weird state where at a random moment a few minutes after startup the laptop slowly loses various functions until it reaches a state where it is irresponsive to any input and must be hard-rebooted, thus rendering it practically useless. As such, Ray Hua made the decision to attempt to live this past semester exclusively off of Athena computing. Hence, why we cared about Athena.)

Programming with `node.js` on Athena is problematic. Specifically, some part of the configuration of Athena makes it absolutely incompatible with `npm`, the node.js package manager. We've tried completely purging all node packages from Athena and reinstalling from scratch. This error still persists.



```
dzaefn@otis-oracle:~$ npm install -g webpack
Unhandled rejection Error: EXDEV: cross-device link not permitted, link '/afs/athena.m
it.edu/user/d/z/dzaefn/.npm/_cacache/tmp/7433acff' -> '/afs/athena.mit.edu/user/d/z/dz
aefn/.npm/_cacache/content-v2/sha512/74/34/a858fabbfa3039eca2870d017b114e25070f01fa35f
d324fdad5aa3c369655263b76ad9c71d5c80276fc0213dff2fd9d2653a38cfd48c2006f1ef999a3eb'

npm ERR! cb() never called!

npm ERR! This is an error with npm itself. Please report this error at:
npm ERR!     <https://github.com/npm/npm/issues>

npm ERR! A complete log of this run can be found in:
npm ERR!     /afs/athena.mit.edu/user/d/z/dzaefn/.npm/_logs/2017-12-05T16_52_56_117Z-d
ebug.log
dzaefn@otis-oracle:~$ npm install -g npm
Unhandled rejection Error: EXDEV: cross-device link not permitted, link '/afs/athena.m
it.edu/user/d/z/dzaefn/.npm/_cacache/tmp/efe8944f' -> '/afs/athena.mit.edu/user/d/z/dz
aefn/.npm/_cacache/content-v2/sha512/29/dd/5ede5ca858f0a2404e52480aceedbdf3f11c2e7f58b
f2280878bfbaaa1b565e5715b9bd7c3ef57442e9c8c158de5be89f025ca81ba16ee6e2ae70a9e992a'

npm ERR! cb() never called!

npm ERR! This is an error with npm itself. Please report this error at:
npm ERR!     <https://github.com/npm/npm/issues>

npm ERR! A complete log of this run can be found in:
npm ERR!     /afs/athena.mit.edu/user/d/z/dzaefn/.npm/_logs/2017-12-05T16_53_09_003Z-d
ebug.log
```

An internet search reveals a newer version of npm fixes this error, but installing the newer version of npm is also made impossible by the error itself. As mentioned above, we attempted completely purging node from a machine and reinstalling from scratch, and for some reason, on Athena this installs version 5.5.1, and not the newer version with the problems fixed.

Due to this, running webpack on Athena was impossible, making testing one's node.js code in Athena substantially more challenging. As such, we resolved to divide up work such that Nchinda did most of the coding and Ray Hua did most of the writeup.

### 4.2 Visual Limitations

We originally intended to visualize a summary-and-cluster view of a van Emde Boas Tree, with the bitvector on the bottom and depictions of the recursive tree above. Eventually, we realized that any reasonably-sized tree would be way too ridiculous for a reasonable screen, and thus opted to visualize an x-Fast Tree instead.

## 5 Links

The visualization itself is available at `http://bst.mit.edu`, hosted via github.io. We plan to transition hosting of this to Scripts, a hosting service run by MIT's Student Information Processing

Board.

The project is entirely free software; the source code is available on GitHub, at https://github.com/Firescar96/geometric-bst-visualizer.

This visualization is linked to on the Wikipedia article for Geometry of Binary Search Trees, at https://en.wikipedia.org/wiki/Geometry_of_binary_search_trees.

## References

[1] Cormen, Thomas, Charles Leiserson, Ronald Rivest, Clifford Stein, *Introduction to Algorithms* (3rd ed.). 2009.

[2] Demaine, Erik, Dion Harmon, John Iacono, Daniel Kane, Mihai Pătraşcu, "The Geometry of Binary Search Trees". *SIAM*, 2009. http://epubs.siam.org/doi/pdf/10.1137/1.9781611973068.55.

[3] Munro, J. Ian, "On the Competitiveness of Linear Search". *ESA*, 2000. https://link.springer.com/content/pdf/10.1007/3-540-45253-2.pdf#page=350.

[4] van Emde Boas, Peter, Robert Kaas, and Erik Zijlstra. "Design and implementation of an efficient priority queue." *Mathematical systems theory*, 1976. https://link.springer.com/article/10.1007/BF01683268.

[5] Willard, Dan E. "Log-logarithmic worst-case range queries are possible in space $\Theta(N)$." Information Processing Letters, 1983. http://www.sciencedirect.com/science/article/pii/0020019083900753.