CrossChx Summary

The following is a list of key points I felt would be worth knowing as they can be more subtle features of the framework that can be creatively flexed in future use cases. Wasn't able to get a computational version of the shortest path, but commented within the code is an attempt. Was however able to code in the ability to directly read in a properly formatted CSV.

Interpreted that shortest route was only via the paths given as these are railroad paths.

Was attempting to use Dijkstra's Algorithm to highlight the shortest path from Frolia to Poipu.  
Tried running a nested for loop to dive through a tree of possibilities. Would require time to research how to do trees within Javascript and apply a recursive function to them. In order to at least get the visual aspect of the stretch goal, manually wrote out the tree and found the path to highlight.   
  
To get the third part of the second stretch goal it wouldn't be too hard to implement into this code. Assuming the function was created and could be called with the start and end index. It would be easy to just read the 0 and 1 spots of the activeNodes array. The shortest path function would likely store the nodes in an array. By pulling the nodes, the IDs on the links would allow easy highlighting of the path. Would probably be harder than it seems applying labels directly to the links. Would be rather easy to update them somewhere on the page outside the SVG though. Wish there was more time to attempt the computational shortest path. Below are the notes mentioned above.

Cx and x attributes serve the same purpose except cx is for circles within svg objects.

dx and dx are just naturally interpreted as the difference based on the nodes location. If an object has an x and y the dx and dy will offset the appearance of the object. Excellent for labels.

cy and cx are absolute coordinates within the svg plane. cx is left to right and cy is top to bottom

nodes inherently hold data x and y which can be called via function(d) { d.x (or d.x)} . This allows them to be bound to attributes which pin them in the correct location on the SVG object.

https://www.dashingd3js.com/using-the-svg-coordinate-space

Using append.append.append will keep adding children to nodes/links ( <g> <circle><text>text</text></circle></g>). .append explicitly several times will lead to several children to a parent object.

Would seem that using a <g> as a node allows other children of any type to be attached. These children seem to inherit the \_\_data\_\_ object specific to that node. This object doesn't seem accessible unless performing d3 functionality. Aka it didn't work when using an .each(function() {})

tick versus end within the function force.on determines if it displays the layout every time a calculation ticks or just at the end of all calculations (aka when everything would come to rest using a "temperature" function).