Tree properties:

* Each node/link is a d3 object, for which an SVG element is displayed
* Tooltips are divs that are positioned next to their node SVGs (divs to allow Latex compilation)
* Nodes
  + Store \_id field of parent and child nodes (along oriented links) in parentsIx and childrenIx arrays
* Links

tree\_of\_knowledge.html

* only “graph” template in the body
* includes all titles, buttons, MathJax call and graphSVG in a div “canvas”

tree\_of\_knowledge.js

* graph loading, re-loading, creating, deleting, etc. functionality
* different graphs are all stored in one DB on the server, with each link/node having a field “graph” that identifies which graph it belongs to.
* Current graph is stored in a Session var
* creates “notify” global function for red in-page notifications
* publishes server DB

tree\_of\_knowledge.css

dbServer.js

* updateNode, updateLink: updates the current db entries by looping over all the fields in the provided objects

tok.js

* nodeData and linkData – array containing all the data for links and nodes [returned or set by force.nodes() ]
* node and link – d3 selections of all nodes and links in the graph
  + the datum of these d3 objects is linked by reference to nodeData and linkData arrays
* use existence of field “source” of linkData to identify it as a link vs. a node
* run button to keep simulation going
* tick: each time-step, update node positions and SVG objects (most forces are implemented by hand here – except for charge repulsion):
  + define a gravity force independent of charge
  + give noise to the nodes to have annealing-like relaxation
  + define soft-max orienting forces for oriented links
  + define the link constraints
  + position all points along the links (3 pts)
  + position each node “group” (tooltip positioning separate) – derivation triangles (need to orient) and other sparately
* redraw: create a visualization from the data
  + set link strengths and node charges and other visual attribudes (position, line type, etc)
* updateSelection – update CSS classes, both for selected and edited link/node
* Layout math:
  + MetaMath: node.importance = 4\*log(num of citations + 1); link.strength = (target node importance + source node imp) / 20;
  + Node.importance = radius in px (scale up template shape); Link.strength = stroke-width in px
  + Node forces:
    - Node.charge = -(importance)^(p+1); Node.chargeDistance = importance \*cnst
    - Charge implemented asymmetrically – view as acceleration
    - (supposedly p=1 here, but empirically p=2 seems more like it..)
  + Link forces: (g=30\*alpha ~O(1))
    - , so adjusting current position automatically changes velocity as well
    - “spring” force:

* + - Orienting force: rotate link by angle

per tick

* + - To get new position, must divide each step by own charge to get the acceleration rather than force (charge = mass here)
  + Scale-invariant dynamics under zoom in/out:
    - If all distances are scaled by factor b (i.e., nd.importance and lk.strength are as well), then to get the same graph layout, we must scale all the couplings as:

Charge=mass:

Links:

* + - Orientation:
  + Remaining free parameters: choice of node importance (radius) and link strengths (widths); transDist; link short and long strengths; orienting force; charge (at equilibr. degenerate with link strengths); charge distance; gravity

tok.css

* Tooltip structure (outer and inner boxes)

gui.js

* All functions for gui operations – keydown, mousedown, etc.
* gui.selected – data array for the selected node (linked by reference)
* showEditor
* all mouse interaction functions

popups.html

* Popup templates

popups.js

* Create and manage editing and display popups
* Update DB: update the database for dat.node according to current form field values; redraw tree and redesplay content, selecting the updated node
  + Takes values from popup fields, does not loop over the values in the linkData array

popups.css