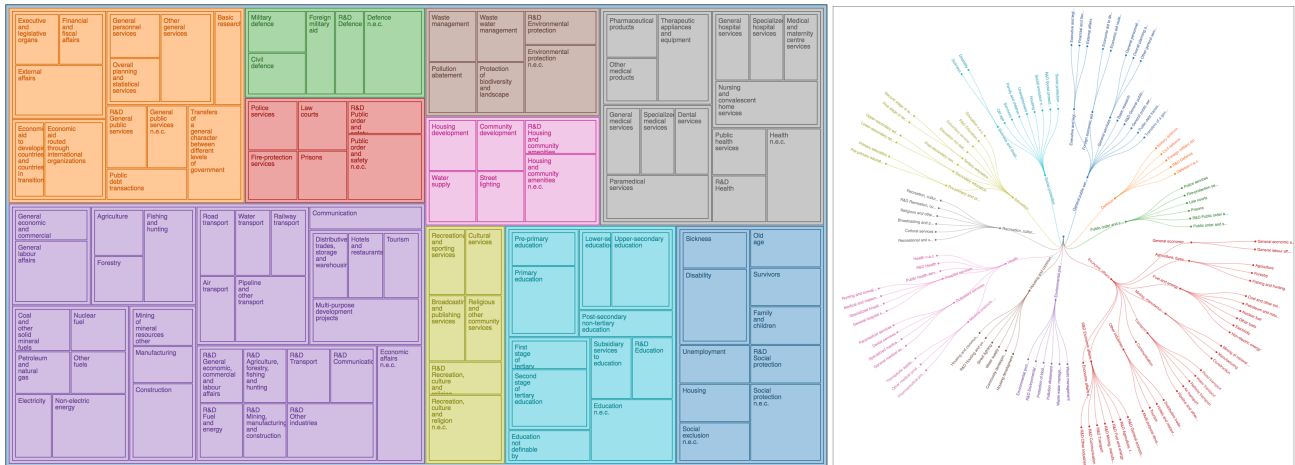


INF552 (2023-2024) - PC s08

Goal: visualize government expenditures (France, 2009) by function, organized into a hierarchy, using D3.
COFOG: Classification Of the Functions Of Government.

We will draw this hierarchy in two ways: as a treemap, and then as a node-link diagram with a radial layout



1. Data Structure

We are only loading one file this time: `cofog.csv`

The data comes as a basic CSV table that does not explicitly encode the hierarchical structure:

```
Level,Code,Amount,Description
1,GF01,,General public services
2,GF0101,,Executive and legislative organs, financial and fiscal affairs, external affairs"
3,GF010101,24.5,Executive and legislative organs
3,GF010102,12,Financial and fiscal affairs
3,GF010103,10,External affairs
2,GF0102,,Foreign economic aid
3,GF010201,2.0,Economic aid to developing countries and countries in transition
3,GF010202,1.5,Economic aid routed through international organizations
...
```

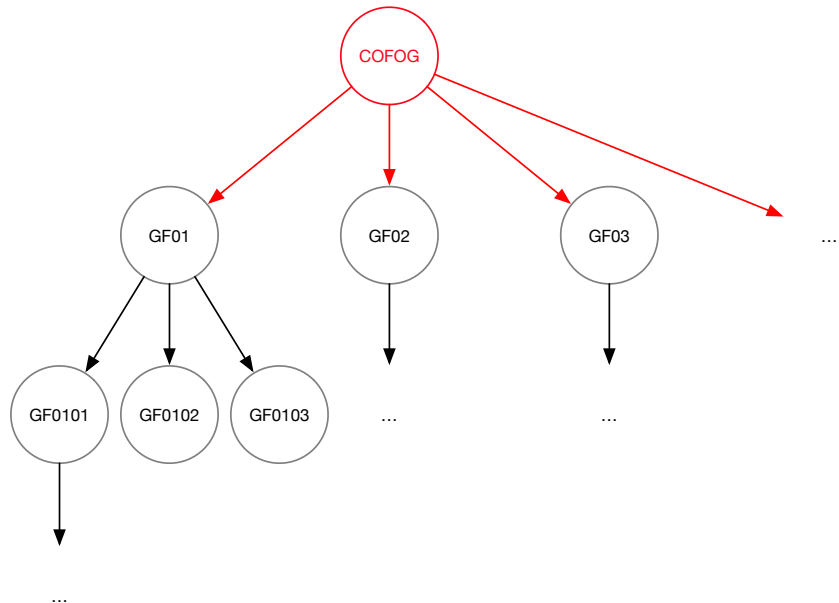
The first task consists of reconstructing the hierarchy and putting it in an instance of `d3-hierarchy` that will later be fed to `d3.treemap()`.

We typically use `d3.stratify()` for this, as in the following example:

```
// if we had the parent explicitly encoded in another column:
let csvData = [
  {Code: "GF01",    parentCode: ""},
  {Code: "GF0101",  parentCode: "GF01"},
  {Code: "GF010101", parentCode: "GF0101"},
  {Code: "GF010102", parentCode: "GF0101"},
  {Code: "GF010103", parentCode: "GF0101"},
  {Code: "GF0102",  parentCode: "GF01"},
  {Code: "GF010201", parentCode: "GF0102"},
  ...
];
// then the call to stratify would be as simple as this:
let root = d3.stratify()
  .id(d => d.Code)
  .parentId(d => d.parentCode)
  (csvData);
```

We do not have anything like `parentCode` in our case, unfortunately. But it is very straightforward to adapt the arrow function that returns the appropriate value to `parentId(...)` based on the observation that in column `Code`, the hierarchy is implicitly encoded as follows: `GF0101xx` are children of `GF0101`; `GF01xx` are children of `GF01`, and so on. Initialize `d3.stratify()` so that the arrow function passed to `.parentId()` returns the proper `parentId` given a `Code` (accessed through argument `d`) simply by truncating it.

Then, before calling `d3.stratify()` on the data, we still need to do one more thing: insert a dummy root node, which we will call `COFOG`:



Indeed, `GF01`, `GF02`, *etc.* need to have a parent node, and that parent being the tree's root, it has to be unique. Just add one more row to the input data array parsed from the CSV file, *before* actually calling `d3.stratify()`.

Node `COFOG` has no parent of its own: be sure to handle this case in the anonymous function that feeds values to `parentId(...)`. Return `null` in that case.

2. Treemap

2.1. Basic Black & White Treemap

Now that we have a proper `d3-hierarchy`, we can initialise the treemap:

```
let treemap = d3.treemap()
  .tile(someTilingMethod)
  .size([w,h]);
```

<https://d3js.org/d3-hierarchy/treemap>

Use `d3.treemapBinary` as the tiling method (works best in this case).

Specify the treemap's dimensions (fill the entire SVG canvas).

Before calling `treemap()` on our data hierarchy, we need to do some more pre-processing on it:

```
root.eachBefore(d => d.data.id = d.data.Code);
root.sum(sumByCount);

function sumByCount(d) {
  return 1;
};
```

Then we can call:

```
treemap(root);
```

What `treemap()` does:

- it decorates nodes in our data structure with geometrical info (each node gets assigned a position and dimensions);
- but it does **not** generate any SVG element. This remains to be done, using typical D3 code to bind data to marks.

Create rectangles, binding them to nodes in the hierarchy:

```
let nodes = svg.selectAll("g")
    .data(root.descendants())
    .enter().append("g")
    .attr("transform", d => `translate(${d.x0},${d.y0})`)
    .classed("leaf", d => d.children == null);

// each node is represented by a rectangle
nodes.append("rect")
    .attr("id", d => d.data.id)
    .attr("width", ...)
    .attr("height", ...)
    /*...*/
```

<https://developer.mozilla.org/en-US/docs/Web/SVG/Element/rect>

Comments about the above code fragment:

- since there is an affine transform applied to `<g>` elements, there is no need to specify (x,y) coordinates on `<rect>` elements;
- class `leaf` is assigned to leaf nodes (necessary to ensure text elements get clipped properly).

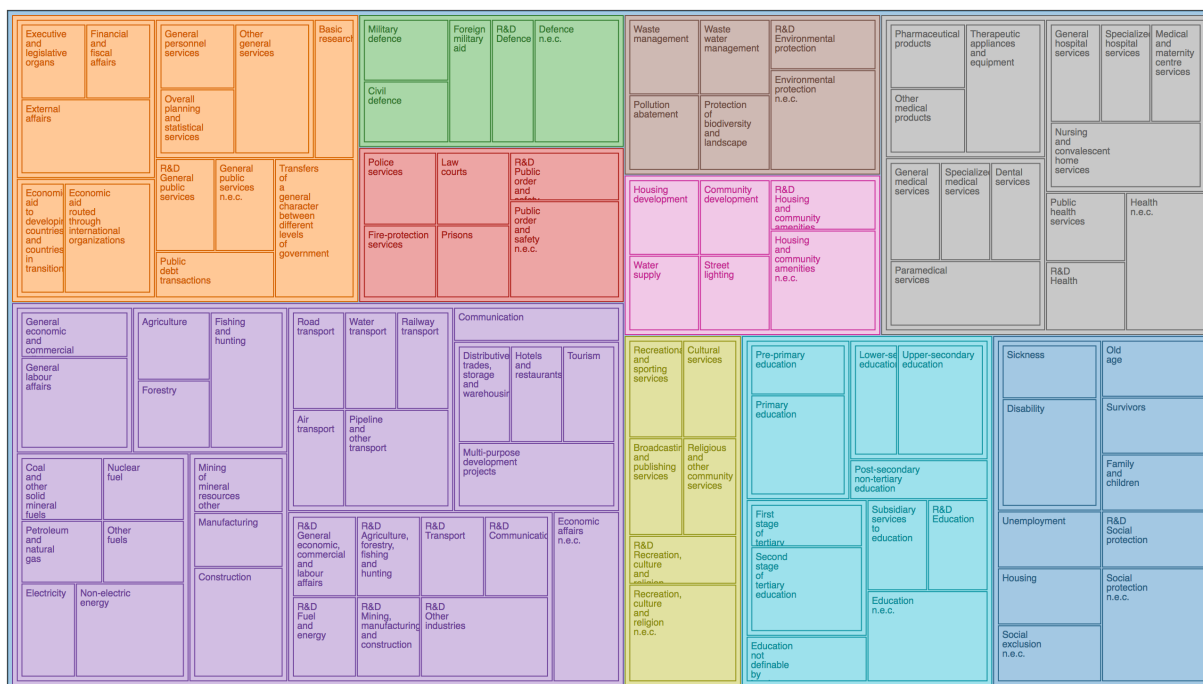
Then we add labels, but only for leaf nodes:

```
d3.selectAll(".leaf").append("text")
    .style("fill", "black")
    .selectAll("tspan")
    .data(d => d.data.Description.split(" "))
    .enter().append("tspan")
    .attr("x", 4)
    .attr("y", (d, i) => (13 + i * 10))
    .text(d => d);
```

Executive and legislative organs	Financial and fiscal affairs	General personnel services	Other general services	Basic research	Military defence	Civil defence	R&D Defence	Waste management	Waste water management	R&D Environmental protection	Pharmaceutical products	Therapeutic appliances and equipment	General hospital services	Specialized hospital services	Medical and maternity centre services
External affairs		Overall planning and statistical services			Foreign military aid		Defence n.e.c.	Pollution abatement	Protection of biodiversity and landscape	Environmental protection n.e.c.	Other medical products				
Economic aid to developing countries and countries in transition	Economic aid routed through international organizations	R&D General public services	General public services n.e.c.	Transfers of a general character between different levels of government	Police services	Fire-protection services	R&D Public order and safety	Housing development	Community development	R&D Housing and community amenities	General medical services	Specialized medical services	Dental services	Nursing and convalescent home services	
		Public debt transactions			Law courts	Prisons	Public order and safety n.e.c.	Water supply	Street lighting	Housing and community amenities n.e.c.	Paramedical services			Public health services	Health n.e.c.
General economic and commercial affairs		Agriculture	Fishing and hunting	Road transport	Water transport	Railway transport	Communication	Recreational and sporting services	Cultural services	Pre-primary education	Lower-secondary education	Upper-secondary education	Sickness	Old age	
General labour affairs		Forestry								Primary education			Disability	Survivors	
Coal and other solid mineral fuels	Nuclear fuel	Mining of mineral resources other than		Air transport	Pipeline and other transport		Multi-purpose development projects	Broadcasting and publishing services	Religious and other community services		Post-secondary non-tertiary education			Family and children	
Petroleum and natural gas	Other fuels	Manufacturing						R&D Recreation, culture and religion n.e.c.		First stage of tertiary education	Subsidiary services to education	R&D Education	Unemployment	R&D Social protection	
Electricity	Non-electric energy	Construction								Second stage of tertiary education	Education n.e.c.		Housing	Social protection n.e.c.	
					R&D Fuel and energy	R&D Mining, manufacturing and construction	R&D Other industries			Education not definable by level			Social exclusion n.e.c.		

2.2. Colored Treemap

We now want to obtain:



Color all nodes based on the 1st-level category (GFnn) using an appropriate color scale from

<https://d3js.org/d3-scale-chromatic>

If need be, desaturate the color palette:

```
let fader = function(c){return d3.interpolateRgb(c, "#fff")(0.6);},
    color = d3.scaleXXX(d3.somePredefinedColorScale.map(fader));
```

Use `tinycolor.js` (already imported) to draw labels using the same color as the corresponding rectangle, but darker (instead of black). See API at: <https://github.com/bgrins/Tinycolor/blob/master/README.md>

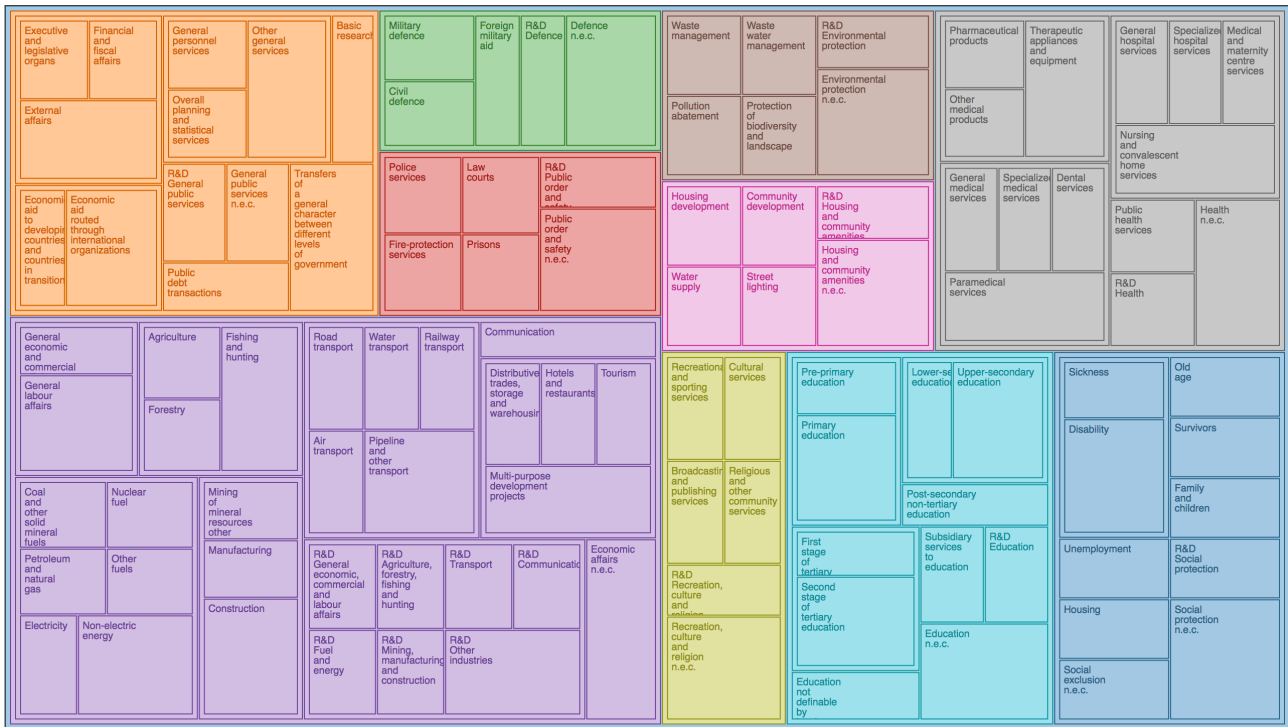
So far, we have achieved this, where we only see leaf nodes:



Adjust `d3.treemap().paddingInner(...).paddingOuter(...)` to explicitly represent the nesting of nodes, effectively showing the hierarchy.

https://d3js.org/d3-hierarchy/treemap#treemap_paddingInner

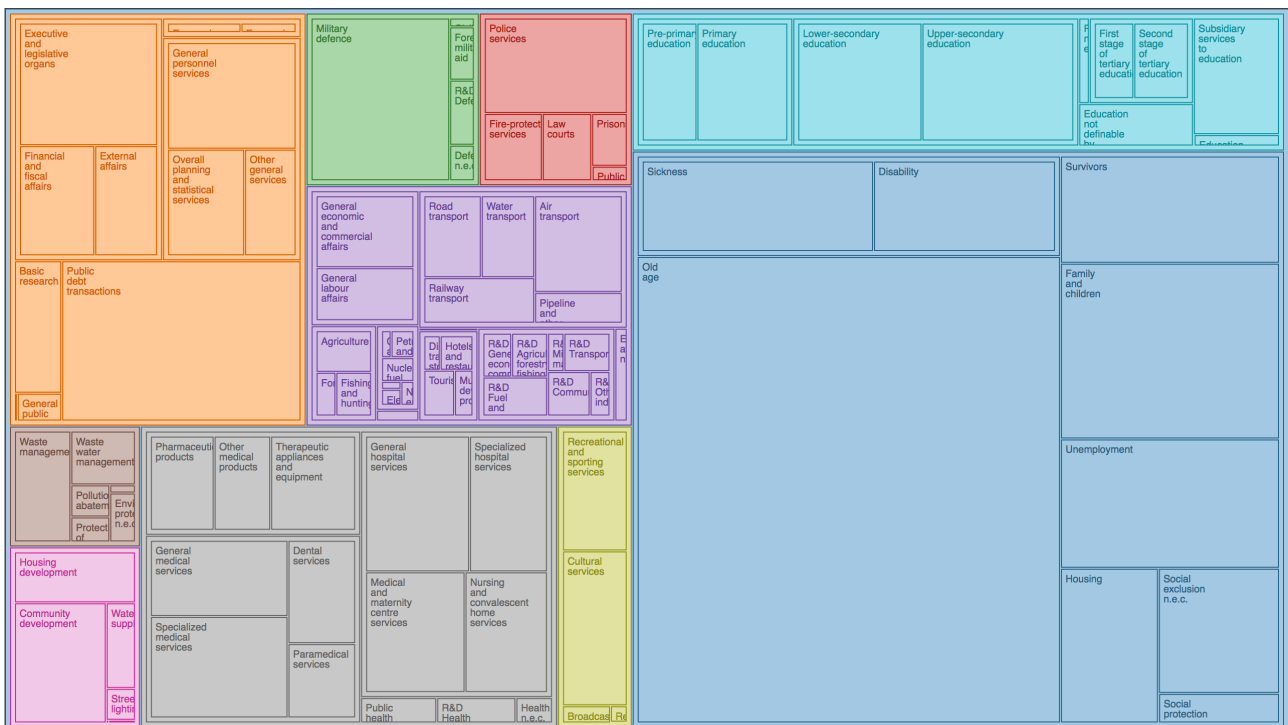
This yields something like:



2.3. Alternative Treemap Layout (optional)

Column Amount in the input CSV holds data about the actual expenditures for each category, in billions of € [Source: National accounts - 2010 base, Insee]

Adapt the treemap layout so that the size of nodes reflects the relative amount of money in each category, as detailed next:



This literally requires changing one line of code:

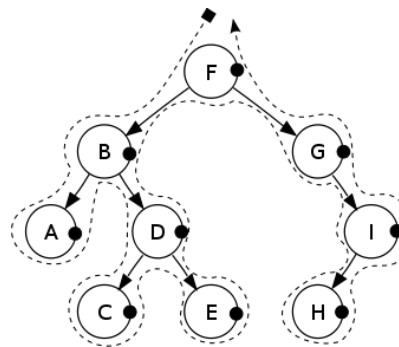
```
root.sum(sumByCount);  
function sumByCount(d){  
    return 1;  
};
```



```
root.sum(sumByAmount);  
function sumByAmount(d){  
    return ...;  
};
```

https://d3js.org/d3-hierarchy/hierarchy#node_sum

Hint: what you have to do is populate each node in the tree with the corresponding `Amount` for leaves, and with the sum of all children's `Amount` for intermediate nodes. Leaves already have their attribute accessible from argument `d`, and given that `node.sum(...)` already performs a *post-order traversal* of the tree, there is really not much to do in function `sumByAmount(d)`!




Last tweak: append a `<title>` to each node's `<g>` element so that the label of non-leaf nodes (intermediate levels in the hierarchy) can also be read when users dwell on them.

3. Node-link Diagram (optional)

We are now switching to
ex08b.html + ex08b.js

Take inspiration from:

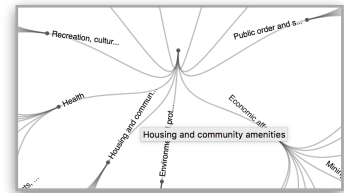
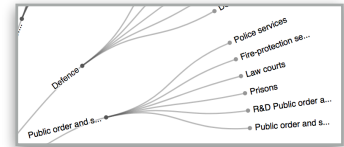
radial_tree_example.html
(in the ZIP file) to create a radial
tree layout 
of the same COFOG data as
used in the treemap.

Reuse the exact same data structure as constructed for the treemap with `d3.stratify()`.

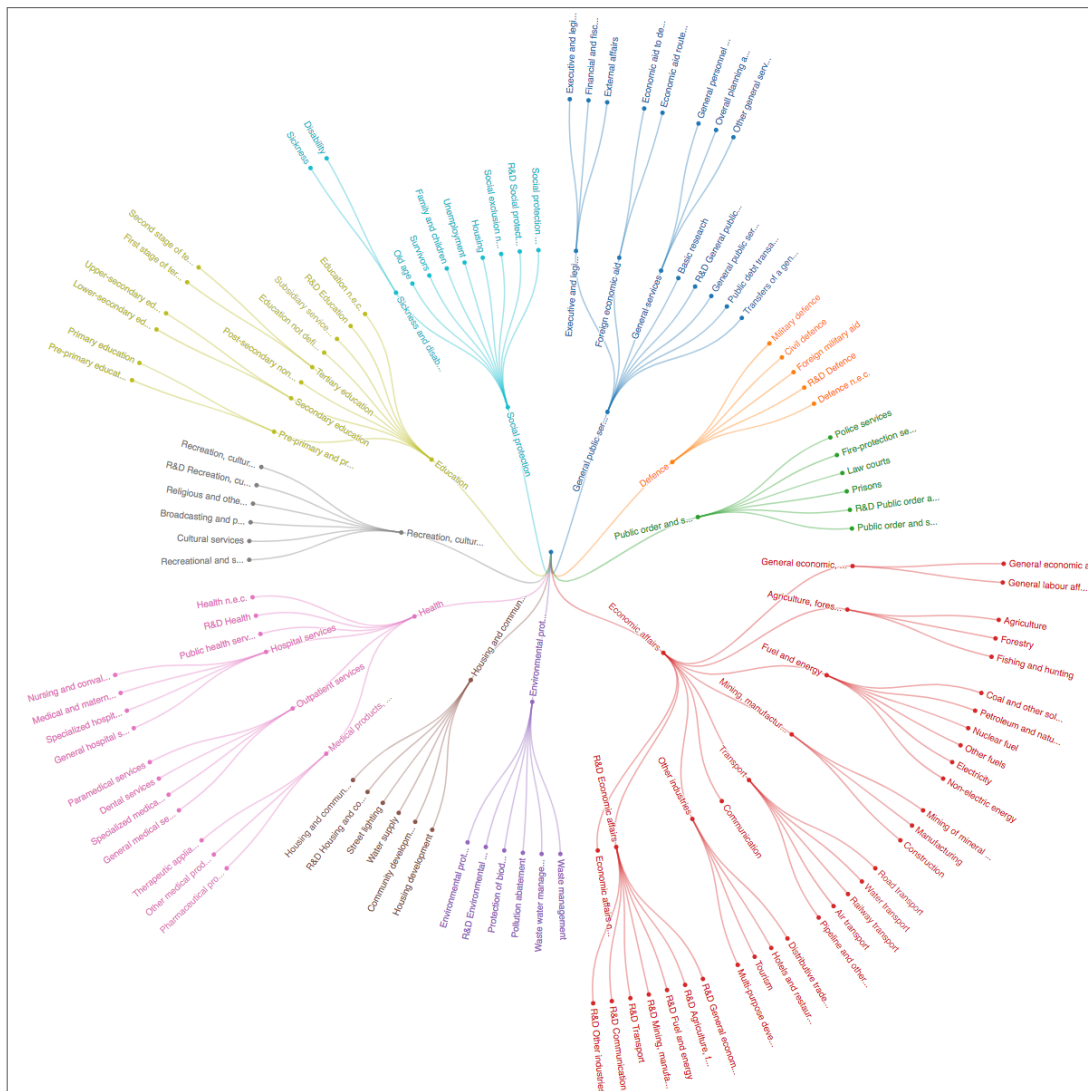


Minor adjustments to the rendering:

- Some labels are too long and will overlap: trim them to 20 chars, append "..." to those that have actually been trimmed.
- As in the previous exercise, append a <title> to each node's <g> element so that trimmed labels can be read in their entirety. This will also make reading labels with a rather vertical orientation easier.



Color all branches based on the 1st-level category (GFnn).



That's just 3 very similar lines of code in the right place... and looking much like those in ex08a (no need for darkening labels here though).