Cápsula 4: *Idioms* jerárquicos en D3.js

Hola, bienvenidxs a una cápsula del curso Visualización de Información. En esta hablaré sobre generación de *idioms* jerárquicos en D3.js.

Al contar con datos en nuestro programa con forma de jerarquía ya, ahora solo queda determinar cómo representarla visualmente y determinar geometrías y posiciones de los elementos gráficos a usar.

D3.js provee varias posibilidades de *idioms* para jerarquías que funcionan de manera similar: crean un objeto generador especializado a una forma de representación visual específica, que recibe una estructura de árbol ya procesada por “d3.hierarchy” o “d3.stratify”; y realiza los cómputos necesarios para posicionar geométricamente cada nodo.

Veremos cuatro casos en esta cápsula, y todos siguen esta idea. Crearemos un generador para un *idiom* específico, le entregaremos los datos jerárquicos ya cargados, y con los resultados crearemos elementos visuales que sigan las coordenadas y geometrías cargadas.

Comenzaremos por un *idiom* de enlace-nodo, llamado internamente por D3.js como “cluster”. Este subdivide un espacio bidimensional rectangular para hacer caber todos los nodos de la jerarquía como marcas de punto con posiciones estratégicas, de tal forma que al conectarse mediante enlaces no existan cruces entre estas.

Para eso, podemos llamar a “d3.cluster” que retorna el generador correspondiente. Que en este caso, especificamos mediante el método “size” el tamaño disponible para distribuir espacialmente, con dimensiones rectangulares. Como en este ejemplo se incluyen márgenes, le restamos el margen del tamaño del SVG completo.

Si aplicamos el generador sobre nuestra jerarquía e imprimimos todos los nodos después de esto, podemos ver los efectos resultantes. A cada nodo se le agregaron atributos “x” e “y”, que reflejan la división espacial producida por el generador en base a la forma jerárquica. Si pensamos en la idea de puntos y enlaces, estos atributos corresponden a las posiciones de los nodos en el fondo.

Para ver efectivamente estas posiciones, agregaré círculos y texto que muestran visualmente las posiciones y el dato correspondiente a cada punto. Para eso, se agregan círculos para cada nodo. Cada uno sigue las coordenadas del nodo, y le aplicamos un radio arbitrario.

En el caso del texto, es similar. Algo a notar es que el texto era parte del objeto cargado originalmente, del archivo, y lo que retorna nuestra objeto jerarquía son nodos, que contienen

ese dato. Entonces es necesario acceder al dato primero, dentro del nodo, y luego al texto que buscamos mostrar.

Si lo probamos, ¡vemos puntos y letras de los nodos! Sin enlaces tal vez no se puede apreciar el orden jerárquico, pero al menos podemos identificar que el nodo raíz “A” está en lo más alto, y luego le siguen sus nodos hijos directos: “B” y “C”.

Ahora agregaremos los enlaces, que crearemos como elementos “path”. Como se desea un camino por enlace, se realiza un *data join* con el arreglo de enlaces de la jerarquía. Para determinar el punto inicial y final de cada uno, puede ser necesario un poco de cálculos geométricos en base a los datos entregados.

En este ejemplo simplifique esta generación con otra función de utilidad de D3: “d3.linkVertical”, que es parte del paquete “d3-shape”. Este provee un generador de líneas en base a datos de enlace. Te invito a revisar los detalles de este generador en la documentación.

Si lo probamos, ¡vemos el resultado! Se genera la jerarquía según el orden especificado en los datos, y de manera muy organizada. El generador “cluster” tiene la particularidad que alinea todos los nodos hoja de la jerarquía en el mismo nivel y al final, para casos donde se prefiere dar énfasis a los nodos hoja.

Alternativamente está el *idiom* de los árboles “d3.tree” que sigue una idea muy similar a la de “cluster”, pero que alinea nodos según su profundidad en la jerarquía, en vez de alinear todas las hojas. El código en pantalla es prácticamente idéntico al de “cluster”, lo único que cambió fue el generador utilizado.

Si lo probamos, vemos la diferencia. Este tipo de distribución permite apreciar la distribución por niveles en la jerarquía, y a distintas profundidades y distancias entre nodos.

Otro *idiom* posible son los mapas de árbol o *treemaps*. Si recuerdas bien, estos dividen un espacio bidimensional en rectángulos autocontenidos que reflejan jerarquía. Su generación sigue la misma idea que los casos anteriores, pero provee posiciones para rectángulos en vez de puntos.

Otra diferencia es que inherentemente se codifica información ordenada mediante el tamaño de cada uno de los rectángulo, por lo que antes de realizar el cálculo de posiciones, el generador espera que cada nodo tenga un atributo “value” numérico para usar como referencia numérica en los tamaños.

Una opción simple, si no se cuenta con valores numéricos en los nodos, es que se codifique la cantidad de nodos hoja en los descendientes de cada nodo. El método “count” de los objetos jerarquía permiten realizar ese cálculo de forma recursiva sobre los nodos.

Si lo probamos, vemos que a la raíz le asigna valor 12, que es la totalidad de nodos hoja en nuestro ejemplo. En cambio los nodos hoja, tienen valor 1. Si ahora también llamamos al

generador de mapa de árbol con esos valores ya calculados, y agregamos rectángulos para cada uno, veremos que produce.

¡Un *treemap* ordenado! Internamente agregué una escala de color que cuantifica la altura del nodo, con lo que colorea a todos los nodos hoja de la misma forma. Podrás apreciar que efectivamente hay proporcionalidad de tamaños entre nodos hermanos que están en una misma contención.

Pero no se aprecia tanto entre nodos hojas de otras partes de la jerarquía. La razón de esto es en general el espaciado que se le aplicó, en este caso con valor 15, para contar con espacio para etiquetar las secciones. Este generador usa por defecto un método específico para realizar los cálculos, pero es posible entregarle otros que realizan estos cálculos de otra forma. ¡Investiga en la documentación otros y prueba por tu cuenta!

También es posible realizar la codificación de tamaños en base a atributos. En pantalla tenemos una jerarquía con la misma estructura que el ejemplo utilizado, pero que cuenta con un atributo numérico “valor” en todo los nodos hoja. Usaremos este *dataset* para reflejar tamaños en base a ese valor.

Para eso, en vez de “count” se puede llamar a “sum”, que mediante la función de acceso especificada, determina el atributo “value” de los nodos hoja, y para el resto de los nodos suma los valores asociados de sus descendientes.

Si probamos esta versión de la jerarquía, vemos una mejor variedad de tamaños, por los valores entregados.

Finalmente, D3.js provee un generador de empaquetamiento circular que sigue una forma similar al *treemap*, pero para generar círculos. También, como este, es necesario que los nodos tengan un atributo “value” calculado previamente. En este ejemplo de código todo es bastante similar al código de *treemap*, pero en vez de rectángulos, se generan círculos.

Si lo probamos, ¡obtenemos el resultado de círculos! Como mencionado en las cápsulas anteriores, se aprecia mejor los distintos niveles de organización y jerarquía, pero el uso de espacio es mucho más ineficiente.

Y con eso termina el contenido de esta cápsula. Te compartiremos el código para que puedas revisarlo en detalle. Recuerda que si tienes preguntas, puedes dejarlas en los comentarios del video para responderlas en la sesión en vivo de esta temática. ¡Chao!