Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Tarea 2

IIC2026 – Visualización de información
Entrega: 21 de octubre de 2018, 19:59

Esta tarea pretende profundizar los conceptos de visualización y D3 aprendidos durante las últimas semanas en clases y ayudantías. En especial abordaremos la representación de grafos. Esta tarea es **individual** y se entrega en su repositorio creado a partir de la siguiente **invitación**.

1. Visualización de dependencias de módulos de D3.js [3 puntos]

1.1. Grafo de dependencias

Como ya se habrán dado cuenta, **D3.js** es una librería altamente modularizada donde cada módulo reúne un set de funcionalidades distintas que ayuda a componer la alta capacidad que tiene la librería para poder manipular documentos en base a datos. A modo de entender de mejor manera el conjunto de herramientas con que podemos disponer **D3.js**, para esta tarea se pide generar un grafo dirigido $G = (V, E)$ de las dependencias de **D3.js**. Cada nodo $v \in V$ representa un módulo de **D3.js** y las aristas representan la relación de dependencia: si v tiene como dependencia a u entonces $(u, v) \in E$. En Siding se encuentra disponible un archivo **JSON** con esta relación, su contenido se extrajo del **repositorio público** de **D3.js**. La estructura del *dataset* es la siguiente:

- El atributo `nodes` contiene los distintos módulos, se especifica para cada uno:
  - `name`: nombre de cada módulo
  - `forks`: cantidad de forks del módulo en GitHub
  - `stars`: cantidad de estrellas del módulo en GitHub
- El atributo `links` contiene las distintas dependencias, se especifica para cada una:
  - `source`: nombre del módulo del cual se depende
  - `target`: nombre del módulo que depende de `source`
  - `dev`: booleano que indica si es una ***devDependency***

1.2. Características

Se debe cumplir con las siguientes características para el grafo:

- Cada módulo debe ser representado por un nodo.
- Cada arista debe indicar claramente cual es la relación de dependencia y su dirección.
- Cada nodo debe mostrar, de alguna forma, los atributos `name` y `forks` del módulo que representa.
- Se debe utilizar algún canal que represente la cantidad de `forks` del módulo que representa.
- Se debe utilizar algún canal sobre las aristas para diferenciar entre una *dependencies* regulares y una ***devDependency***.
- Al pasar el cursor sobre un nodo (i.e. ***hover***), la visualización debe realizar énfasis sobre ese nodo, sobre los nodos relacionados (o vecinos) y las aristas que los conectan.
- Debe poder arrastrar cada nodo con el cursor (i.e. ***drag***).
- Al arrastrar un nodo, su posición debe quedar fija incluso si se deja de arrastrar. Sólo si se hace **clic** (i.e. ***click***) sobre el nodo, se libera su posición.
- Los nodos no pueden salir del marco donde se visualiza el grafo, aunque se intente arrastrar fuera de ella.
- Al hacer **doble clic** (i.e. ***dblclick***) en un nodo, también se desencadena una actualización en el ***barchart*** de la sección 2 con la información del nodo cliqueado y sus vecinos.

2. Comparación de popularidad entre los módulos [1.5 puntos]

Gracias al grafo podemos determinar las relaciones de dependencia que existen entre los determinados módulos de **D3.js**, ahora bien, no es fácil determinar cual son los módulos más populares. Podríamos inferir que existe una relación lineal entre la cantidad de `forks` y `stars` en GitHub, pero esto puede no ser siempre verdad. Para esto se pide confeccionar una comparación mediante un `barchart` que permita apreciar la diferencia de popularidad entre módulos vecinos.

2.1. Características

- El largo de las líneas debe codificar la cantidad de `stars` del módulo.
- Cada vez que se seleccione un módulo junto a sus vecinos, se deben actualizar las escalas de los ejes de manera que los largos de las líneas se ajuste al nuevo conjunto de módulos mostrados.
- Debe existir un botón que elimine todos los elementos del ***barchart***.
- Debe mostrarse de alguna forma los nombres de los módulos comparados.

- Se debe utilizar algún canal dentro del `barchart` para codificar al módulo al que se le hizo **doble clic** en el grafo.

3. Presentación de resultados [1.5 puntos]

3.1. Informe

Debe crear un informe donde presente los resultados obtenidos en esta tarea. Este debe ser un documento **HTML** que explique los siguientes puntos:

1. Explicación paso a paso de como se genera cada gráfico. Esto es desde que se abre el archivo hasta que se obtiene el resultado final. Considere que debe haber una explicación distinta por gráfico puesto que ambos buscan distintos objetivos. Bastan con explicaciones de alto nivel, no es necesario explicar cada línea de código, pero tampoco ser tan alto nivel como: "Incluí los datos a un *barchart* con D3 y se generó el gráfico".
2. Se se espera que todas las decisiones de visualización —entiéndase como toda decisión de *visual encoding* tomadas en ambas visualizaciones— estén fundamentadas correctamente en base a lo visto en el curso.

Recuerde hacer uso de los diferentes *tags* que posee **HTML** para mostrar la información de forma ordenada y amigable para el lector. Se espera que al menos el documento contenga lo siguiente:

- Contenido centrado (textos e imágenes) y texto justificado.
- Cambios adicionales a un archivo **CSS** correspondiente, que enriquezcan la presentación de su informe.

No olvide que este documento será su informe de la tarea, por lo que adicionalmente se evaluará que cumpla con una correcta redacción y ortografía. Puede incluir un archivo `README.md` si es necesario realizar un paso previo para lograr cargar su visualización, como realizar el comando `python -m http.server 8000`, pero todo lo demás debe estar contenido en su informe **HTML**.

3.2. Formato

En esta tarea debe respetar los siguientes formatos:

- Entregar los dos *scripts* escritos en **JavaScript** en archivos separados.
- Debe utilizar la versión 5 de **D3.js** y programar con un paradigma **declarativo**. Por ejemplo, no se espera ver un `for` para hacer agregar cada elemento **SVG**, sino que utilizar el revisado *data-join* para agregar elementos.

4. Entregables

Usted debe entregar su tarea en el repositorio que se le crea automáticamente al ingresar al **enlace** señalado anteriormente. Los entregables que se esperan de esta tarea son:

- **Presentación resultados**: un informe en formato **HTML** que detalle lo realizado y logrado en las anteriores secciones y un archivo **CSS** que enriquezca su presentación.
- `graph.js`: *script* **JavaScript** realizado con **D3.js** que genere el grafo de dependencias.
- `barchart.js`: *script* **JavaScript** realizado con **D3.js** que genere el *barchart* para comparar popularidades.
- `README.md`: un archivo en formato **Markdown** en caso de detallar alguna instrucción previa para visualizar la tarea. Esto es opcional.

5. Política de atraso

En la eventualidad de entregar pasada la fecha de entrega, se aplicará un **descuento** a la nota final obtenida en su tarea.

De haber atraso, el descuento comienza desde las 5 décimas. El descuento aumenta linealmente hasta las 24 horas posteriores del plazo inicial hasta un total de 20 décimas. Pasado un día del plazo inicial, el descuento es de 70 décimas, es decir, nota final **1**. La figura 3 muestra la función de descuento en función a las horas de atraso.

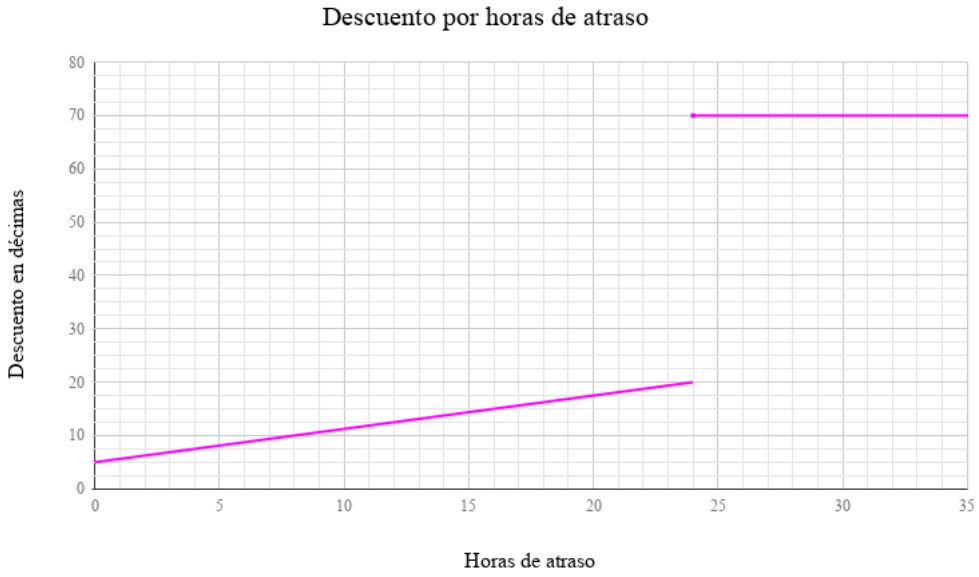

Figura 1: Descuento en nota según horas de atraso.