UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Librería de Visualización de Estructuras de Datos

MEMORIA PARA OPTAR AL TÍTULO DE
Ingeniero Civil en Computación

POR
José Luis Romero Munizaga

PROFESOR GUÍA:
Ivan Sipiran Mendoza

MIEMBROS DE LA COMISIÓN:
Dionisio González González
Federico Olmedo Berón

Santiago de Chile
2022

Resumen

Las estructuras de datos son uno de los temas más importantes en las Ciencias de la Computación. Por el mismo motivo son parte de los contenidos básicos que se enseñan en esta disciplina. Frecuentemente en la docencia se utilizan diagramas de las estructuras de datos como un apoyo al aprendizaje, pero estos diagramas son estáticos y no capturan la naturaleza dinámica de las estructuras de datos. Si bien, si existen visualizaciones animadas de estructuras de datos, estas son poco flexibles y están separadas del ambiente donde se implementan las estructuras de datos.

En particular, para el curso Algoritmos y Estructuras de Datos del Departamento de Ciencias de la Computación de la Universidad de Chile se desarrolló previamente una herramienta que genera diagramas estáticos y existe la necesidad de una herramienta similar, pero que permita generar visualizaciones animadas.

Por lo anterior, sería útil contar con una herramienta que permita generar visualizaciones animadas de estructuras de datos que esté incorporada en el mismo ambiente de desarrollo. De esta manera, el usuario puede visualizar la estructura de datos al mismo tiempo que la implementa, incluso permitiéndole ver cuando su implementación tiene errores.

Para satisfacer esta necesidad se creó *dsvisualizer*, una herramienta para generar visualizaciones animadas de estructuras de datos implementadas en Python. En su versión actual solo permite visualizar listas enlazadas, pero está diseñada para permitir generar visualizaciones de más estructuras de datos en el futuro. Le permite al usuario generar una animación de su propia implementación, minimizando la intervención que debe realizarle a su código.

Para lograr esto, *dsvisualizer* tiene un *back-end* y un *front-end*. El *back-end*, implementado en Python, captura las operaciones sobre la estructura a visualizar y genera un modelo de estas que se lo envía al *front-end*. Este, implementado en TypeScript, recibe el modelo desde el *back-end* y a partir de este genera la visualización animada, utilizando la librería de animación D3js.

Para evaluar la herramienta se hicieron pruebas con usuarios —12 estudiantes de la Facultad de Ciencias Físicas y Matemáticas—, donde se les pidió utilizar la herramienta y luego contestar un cuestionario. La primera parte de este corresponde a la escala de usabilidad SUS, en la cual la herramienta obtuvo un puntaje promedio de 90 de 100 puntos. La segunda parte consiste en preguntas abiertas, donde se preguntó por: comentarios positivos, comentarios negativos y oportunidades de mejora. El resultado de la evaluación fue muy positivo, pero de todas maneras se identificaron varias oportunidades de mejora y nuevas funcionalidades que contribuirían a hacer la herramienta aún más útil.

A los futuros estudiantes de Ciencias de la Computación.

Agradecimientos

Agradezco a Ivan Sipirán, por su excelente disposición, por sus sugerencias y por su apoyo como profesor guía durante todo el proceso de la memoria.

También quiero agradecer a todos los estudiantes que participaron de la evaluación y contribuyeron con su tiempo a mejorar esta herramienta.

Por otra parte, agradezco a mis amigos y a mi familia, por el ánimo, apoyo y ayuda que me han dado siempre y me dieron mientras trabajaba en la memoria.

Tabla de Contenido

Índice de Ilustraciones

Índice de Tablas

Capítulo 1

Introducción

Posiblemente, uno de los temas más relevantes en las Ciencias de la Computación corresponde a las Estructuras de Datos. Las estructuras de datos son maneras de almacenar y organizar los datos para facilitar su acceso y modificación [3]. Comúnmente se utilizan diagramas y otras visualizaciones para facilitar la comprensión de ellas y las operaciones que se realizan sobre ellas. En particular, las visualizaciones de estructuras de datos pueden ser útiles para la docencia en Ciencias de la Computación y pueden ser utilizadas tanto por los estudiantes como por los profesores. Las visualizaciones de algoritmos son efectivas para ayudar en el aprendizaje de estudiantes de Ciencias de la Computación [5].

Tradicionalmente, estas visualizaciones han sido diagramas estáticos, pero como las estructuras de datos son dinámicas, puede ser útil contar con visualizaciones animadas. Además, las visualizaciones animadas suelen estar separadas del ambiente donde se implementan las estructuras de datos y podría ser útil que estas animaciones estén integradas en el ambiente de desarrollo.

El ambiente de desarrollo interactivo Jupyter Notebooks es muy popular y es una buena opción para crear esta herramienta porque permite mostrar elementos interactivos y animados dentro del mismo ambiente de desarrollo. El lenguaje más utilizado en este ambiente es Python, el cual además es el segundo lenguaje de programación más utilizado según una encuesta anual realizada por GitHub, State of the Octoverse [4].

En particular, en el curso CC3001 – Algoritmos y Estructuras de Datos de la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile se enseñan estructuras de datos y se utiliza Python en conjunto con Jupyter Notebooks. Para este curso se desarrolló una herramienta para ayudar en la docencia que genera visualizaciones estáticas. Sin embargo, esta herramienta tiene varias limitaciones y sería útil contar con una herramienta que solucione estos problemas y permita generar visualizaciones animadas.

Para lograr esto se implementó la herramienta *dsvisualizer*, una librería de Python que genera visualizaciones animadas de estructuras de datos implementadas por los usuarios en Jupyter Notebooks. En su versión actual se limita a listas enlazadas, pero está diseñada para poder expandirla a otras estructuras de datos. La herramienta utiliza la librería Jupyter Widgets para poder dibujar las visualizaciones animadas en un Notebook. Tiene un back-end implementado en Python que se encarga de registrar las operaciones realizadas por el usuario y un front-end escrito en TypeScript que genera las visualizaciones a partir de este registro. Las dos partes de la herramienta se comunican usando la

librería ipywidgets, que permite mantener un modelo sincronizado entre el front-end y el back-end.

Para validar la herramienta se realizaron pruebas con estudiantes de la FCFM. Durante estas, los usuarios probaron la herramienta y luego contestaron un cuestionario. La primera parte del cuestionario es el formulario System Usability Scale (SUS) —un método estándar para evaluar usabilidad—, la segunda parte está compuesta por preguntas abiertas sobre la herramienta, y la tercera parte es una caracterización del usuario.

1.1. Objetivos

Objetivo General

Diseñar e implementar una librería para generar visualizaciones de estructuras de datos en Notebooks de Python que sea efectiva, eficiente y usable.

Objetivos Específicos

1. Diseñar e implementar el widget para que sea fácil de usar.
2. Diseñar e implementar el modelo del widget.
3. Implementar la visualización de la estructura de datos a partir del modelo.
4. Evaluar la utilidad y la usabilidad de la librería.

Capítulo 2

Estado del Arte

2.1. Librerías de visualización de estructuras de datos

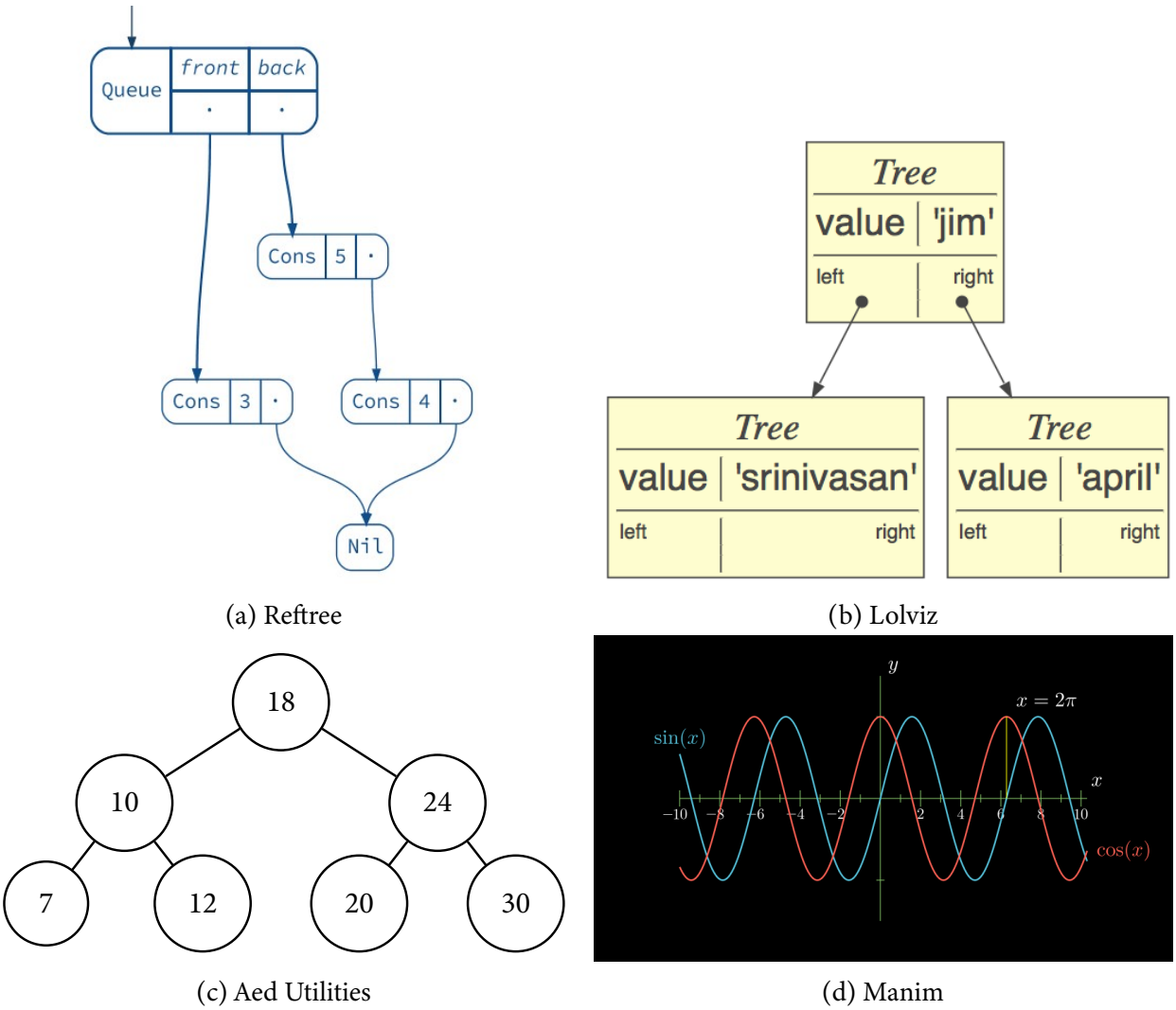
Actualmente, existen varias librerías para generar visualizaciones de estructuras de datos, pero muchas de ellas están implementadas en lenguajes distintos de Python y no están diseñadas para poder ser utilizadas en Notebooks. Un ejemplo de esto es la librería Reftree [13], que permite crear visualizaciones de estructuras de datos, pero en el lenguaje Scala, por lo tanto, no se puede utilizar para visualizar estructuras de datos implementadas en Python y tampoco se puede utilizar en Jupyter Notebooks. Otro ejemplo es la librería Lolviz [9], que, si bien si permite visualizar en Notebooks estructuras de datos implementadas en Python, cuenta con la limitación de que no puede generar animaciones. En la figura 2.1 se pueden ver ejemplos de las visualizaciones generadas por estas librerías.

Por otro lado, existe la librería Aed-Utilities [12] creada para el curso de Algoritmos y Estructuras de Datos de la FCFM por el Profesor Ivan Sipirán; que, si bien permite generar diagramas de estructuras de datos en Python y mostrarlas en un Notebook, cuenta con las siguientes limitaciones: no permite crear animaciones, la API no es muy cómoda de utilizar y el algoritmo que utiliza para generar las visualizaciones es poco eficiente.

Tanto Aed-Utilites como Lolviz generan los diagramas utilizando Graphviz —una herramienta originalmente desarrollada por AT&T para dibujar gráficos especificados en el lenguaje DOT— que permite generar diagramas de muy buena calidad, pero no permite crear animaciones.

En cuanto a herramientas para generar animaciones, existe la librería Manim [14], diseñada para generar visualizaciones animadas de matemáticas. Si bien no está diseñada para crear visualizaciones de estructuras de datos, es relevante porque permite generar visualizaciones animadas y estas se pueden ver desde Jupyter Notebooks. Esta librería tiene un diseño orientado a objetos, donde una animación es un objeto que tiene un campo con el objeto matemático que representa y tiene un método que permite animar el objeto según una función de interpolación. Para generar el resultado final puede utilizar varios back-ends de renderización, incluyendo OpenGL y WebGL. Utilizando este último cuando se usa desde un Notebook. En la figura 2.1d se puede ver una visualización de una función creada con esta herramienta.

Otra herramienta enfocada en las matemáticas es Penrose [15], una herramienta que permite crear diagramas a partir de un programa en un lenguaje específico de dominio. La disposición de los



(a) Reftree

(b) Lolviz

(c) Aed Utilities

(d) Manim

Figura 2.1: Ejemplos de visualizaciones generadas por las distintas herramientas. Obtenidas de [13], [9], [12] y [14], respectivamente.

elementos en el diagrama se obtiene mediante optimización numérica. El lenguaje específico de dominio (o DSL, por sus iniciales en inglés) se separa en un archivo que define solo la parte matemática y en otro que define el estilo.

Dado que no se encontró ninguna herramienta existente que permita visualizar estructuras de datos implementadas en Python, permita generar animaciones y permita mostrar las visualizaciones generadas en un Jupyter Notebooks, se requiere crear una nueva librería que implemente estas funcionalidades.

2.2. Jupyter Notebooks

Los Jupyter Notebooks son archivos interactivos que almacenan bloques de texto, código y también pueden contener imágenes, videos y animaciones interactivas. Un Jupyter Notebook tiene varios componentes que permiten construir la experiencia de usuario. En primer lugar, está el archivo del notebook, que es un archivo JSON con la extensión `.ipynb`, que almacena en el disco todos los datos

persistentes del notebook, incluyendo los bloques de texto, los bloques de código y sus salidas, las imágenes, etc. Después, está el *kernel* o núcleo, que es un intérprete del lenguaje respectivo, con la capacidad de comunicarse con el servidor del notebook. El servidor del notebook se encarga de la comunicación entre el front-end, el kernel y el archivo del notebook. Finalmente, está el front-end del notebook, que se encarga de mostrarle al usuario una representación del notebook y le permite a este interactuar con el notebook, usualmente corresponde a un navegador web. En la figura 2.2 se puede ver un diagrama de esta arquitectura.

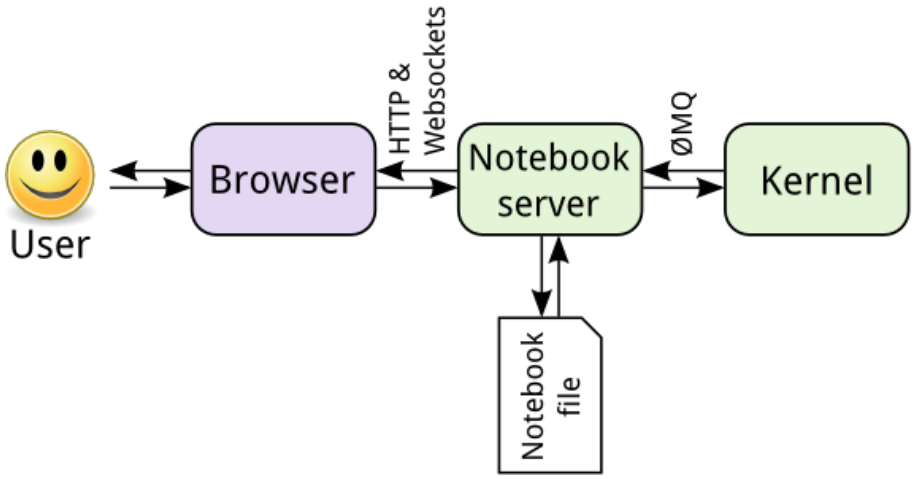


Figura 2.2: Arquitectura de Jupyter Notebooks. Obtenido de [6].

Existen múltiples *kernels*, uno por cada lenguaje que soportan los Jupyter Notebooks, pero el más conocido es el kernel IPython, que es el kernel del lenguaje Python. Este, además de ser un kernel de Jupyter Notebooks, es un intérprete interactivo de Python, con funcionalidades como contenido multimedia y completado de comandos. Cuando se utiliza un notebook con el lenguaje Python, el servidor del notebook se comunica con este intérprete, que es el que se encarga de interpretar los bloques de código y responder con la salida generada cuando el servidor del notebook envía el request.

Para crear componentes interactivos en un Notebook de Python se utiliza la librería Jupyter Widgets. Esta librería permite definir un modelo que representa el componente, y una vista en JavaScript y HTML que es mostrada por el front-end del Notebook. El modelo se define tanto en JavaScript como en Python y la librería se encarga de mantener las dos versiones sincronizadas. Para mantener las dos versiones sincronizadas, es necesario serializar y deserializar los campos del modelo, para poder enviarlos en formato JSON. Entonces, para los campos sencillos, la librería se puede encargar de la serialización o deserialización, pero para los casos más complejos se deben definir manualmente los serializadores y deserializadores para cada campo.

En la figura 2.3 se observa la arquitectura de un widget generado con esta librería. Aquí se puede observar como el usuario interactúa con el navegador, el cual se comunica con HTTP y WebSockets con el servidor del Notebook. Este se comunica utilizando ZeroMQ —una librería de comunicación asíncrona orientada a mensajes— con el *kernel* y se comunica utilizando la interfaz del sistema operativo con el archivo del Notebook.

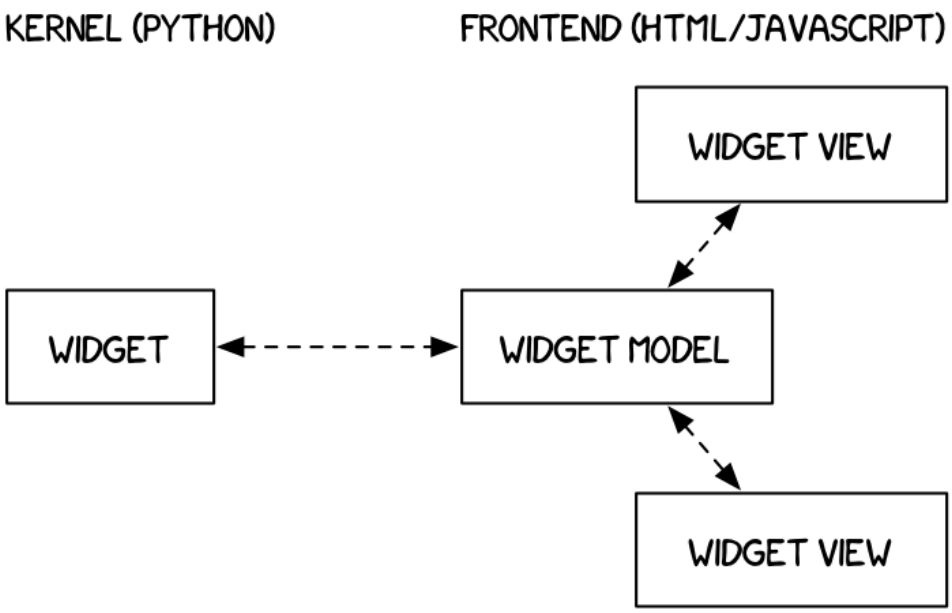


Figura 2.3: Arquitectura de Jupyter Widgets. Obtenido de [7].

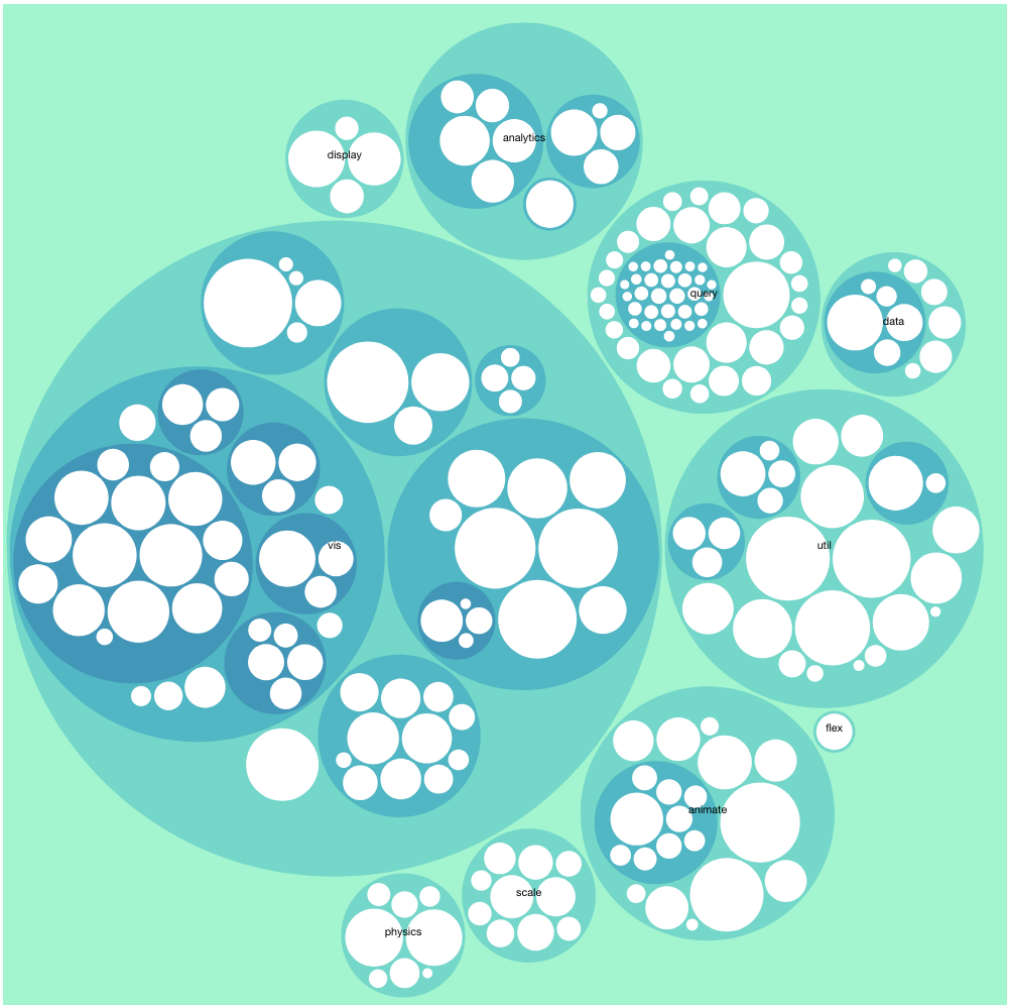


Figura 2.4: Ejemplo de visualización generada con D3

2.3. D3

D3 (Data-Driven Documents) es una librería de JavaScript para manipular documentos en función de datos. Esta librería permite manipular el DOM (Document Object Model) y aplicarle transformaciones según los datos. Permite usar las tecnologías estándar de la web para generar visualizaciones de buena calidad.

Es particularmente útil la capacidad de D3 de manipular elementos SVG (Scalable Vector Graphics) —un lenguaje de markup para describir gráficos en dos dimensiones—. Esto permite generar animaciones en dos dimensiones de forma relativamente simple. Ya que se pueden definir programáticamente los elementos SVG a partir de los datos y la herramienta permite hacer la interpolación entre el estado actual y el estado al que se quiere llegar. Además, esta técnica es bastante eficiente gracias a que los navegadores implementan motores muy eficientes para la renderización de elementos SVG.

Para generar las visualizaciones, D3 permite enlazar datos con elementos del DOM y definir los atributos de estos elementos como función de los datos. Además, cuando cambian los datos permite hacer una transición suave de los elementos correspondientes a su nueva posición según la función definida anteriormente. La figura 2.4 muestra un ejemplo de una visualización generada con D3 que representa un conjunto de datos como una agrupación de círculos. Esto muestra la versatilidad de D3 a la hora de generar visualizaciones, lo cual es una de sus principales ventajas.

2.4. SUS

La escala SUS, o System Usability Scale utilizando su nombre completo, fue introducida en [2] por John Brooke como un método "*quick and dirty*" (rápido y sucio) para evaluar usabilidad, desarrollado en Digital Equipment Corporation (DEC). Desde entonces, se ha vuelto un método estándar en la industria para medir la usabilidad en todo tipo de sistemas. En [1] analizan 10 años de datos de evaluaciones hechas con SUS y encuentran que esta escala es una herramienta altamente robusta y versátil para las evaluaciones de usabilidad.

SUS consiste en un formulario de 10 ítems que se le aplica a usuarios después de que utilizan el sistema que se está evaluando. Idealmente, el usuario debe contestar el formulario inmediatamente después y con sus primeras impresiones en vez de pensar mucho las respuestas. Cada ítem consiste en una afirmación para la cual el usuario debe marcar, en una escala de Likert del 1 al 5, si está muy en desacuerdo o muy de acuerdo con la afirmación. Las afirmaciones están alternadas entre afirmaciones que reflejan de forma positiva y negativa sobre la usabilidad, con el objetivo de obligar a los participantes a pensar sobre la respuesta de cada ítem. Las afirmaciones en la versión original son:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly

8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

A partir de las respuestas del usuario se obtiene un puntaje entre 0 y 100. Para calcular el puntaje, cómo las preguntas están alternadas, primero se deben transformar los puntajes de forma que sean consistentes. Para esto, se transforma el puntaje de cada ítem a una escala entre 0 y 4, donde 0 es lo peor y 4 es lo mejor. Esto se logra restándole 1 a los ítems impares y restando de 5 los ítems pares. Después, sumando los puntajes transformados, se obtiene un puntaje total entre 0 y 40, que se multiplica por 2.5 para obtener un puntaje entre 0 y 100.

En [10] reportan que utilizando datos de 446 estudios y más de 5000 respuestas individuales, encontraron que el promedio del puntaje fue 68 con una desviación estándar de 12.5. Por el mismo motivo, recomiendan transformar el puntaje a un percentil para interpretar los resultados y presentan una tabla para transformar los puntajes a percentiles.

El cuestionario asociado a esta escala está en inglés, por lo que debe traducirse al español para aplicarlo a un grupo cuya lengua nativa es el español. En [11] tradujeron el cuestionario al español, validaron el cuestionario traducido con expertos y luego lo probaron con 88 usuarios. Finalmente, concluyeron que la versión traducida del cuestionario es una herramienta válida y confiable para evaluar la usabilidad de herramientas electrónicas para usuarios hispanohablantes.

Capítulo 3

Problema

Tradicionalmente, las visualizaciones de estructuras de datos que se utilizan son diagramas estáticos. Sin embargo, dado que las estructuras de datos son dinámicas y no estáticas, es útil utilizar visualizaciones que sean animaciones dinámicas de las estructuras de datos. Comúnmente, las animaciones de estructuras de datos son videos o páginas web interactivas, pero en ambos casos, desconectadas de donde se programan estas estructuras de datos. Por lo tanto, sería útil contar con una herramienta para generar visualizaciones animadas de estructuras de datos que se pueda usar desde el mismo ambiente de desarrollo.

En los últimos años se ha vuelto más común la utilización de Jupyter Notebooks en la enseñanza de Ciencias de la Computación. Los Notebooks son ambientes de desarrollo interactivos basados en tecnologías web, que permiten combinar bloques de texto, bloques de código y los resultados generados por estos, para ser mostrados por una aplicación web interactiva [8]. Los resultados pueden incluir texto, imágenes, animaciones y elementos interactivos. Originalmente fueron diseñados para ser utilizados en el ámbito de la Ciencia de los Datos, pero han estado tomando popularidad en otras áreas de la computación además de la docencia.

El lenguaje de programación Python es un lenguaje de alto nivel, multiparadigma, de uso general y usualmente interpretado que se ha vuelto uno de los lenguajes más populares. Según la encuesta realizada anualmente por GitHub, State of the Octoverse, es el segundo lenguaje más popular, solo siendo superado por el lenguaje de programación JavaScript [4]. Además, probablemente es uno de los lenguajes más utilizados en la docencia de Ciencias de la Computación.

Dada la popularidad de los Jupyter Notebooks y del lenguaje de programación Python, especialmente en docencia, sería beneficioso contar con una herramienta para generar visualizaciones animadas de estructuras de datos que se pueda usar en Notebooks en el lenguaje de programación Python. De esta manera se aprovecha que los Jupyter Notebooks, a diferencia de otros ambientes de desarrollo, permite generar elementos interactivos usando las tecnologías web.

En particular, en el curso de CC3001 – Algoritmos y Estructuras de Datos de la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile se utiliza tanto Python, como Jupyter Notebooks y el Profesor Ivan Sipirán, académico de esa facultad, desarrolló una librería para ese curso para generar visualizaciones de estructuras de datos denomianda *aed-utilities* [12]. Esta librería tiene como objetivo de ayudar en la docencia permitiéndole tanto a los alumnos como a los profesores generar visualizaciones de estructuras de datos implementadas por ellos, por el profesor, o por algún

libro o recurso externo. Sin embargo, esta librería actualmente cuenta con una serie de limitaciones: La implementación no es tan eficiente como podría ser, la API (Application Programming Interface) que se utiliza para usar la herramienta no es muy ergonómica y no permite visualizar cómo se ejecutan las operaciones sobre las estructuras de datos (solo permite visualizar una estructura de datos en un momento en el tiempo).

El problema abordado consiste en implementar una herramienta que permita generar visualizaciones animadas de las operaciones que se realizan sobre una estructura de datos implementada por el usuario. Para esto, la herramienta debe ser capaz de registrar las operaciones sobre la estructura y debe ser capaz de generar una visualización animada a partir de las operaciones que registró. Es deseable que la visualización se genere en el mismo notebook y que el usuario tenga que agregar la menor cantidad de instrumentación necesaria a su código para que la herramienta funcione.

Implementar una librería que cumpla con estos requerimientos sería beneficioso para estudiantes de Ciencias de la Computación, profesores, o personas en general que estén aprendiendo estructuras de datos, o que deseen generar visualizaciones de estructuras de datos implementadas en Python y verlas en Jupyter Notebooks. Además, en particular, permitiría contribuir a la docencia en el curso CC3001 – Algoritmos y Estructuras de Datos de la FCFM.

Capítulo 4

Solución

4.1. Arquitectura

La solución es una librería de Python para generar visualizaciones animadas de estructuras de datos y las operaciones que se realizan sobre estas, que pueda ser usada en Jupyter Notebooks.

La librería le permite al usuario implementar una lista enlazada y, agregando la instrumentación provista por la librería, le permite generar una visualización animada de las operaciones que se realizaron sobre la estructura.

Para permitir esto, la librería está compuesta por dos partes: el *back-end* y el *front-end*. Estas dos partes se comunican entre sí utilizando un modelo de datos común que cada una de las partes sabe serializar y deserializar.

El *back-end*, implementado en Python, define la instrumentación para capturar las operaciones realizadas sobre la estructura de datos. Se encarga de mantener un registro de todas las operaciones que se realizan sobre esta. Este registro se mantiene utilizando una representación de las operaciones primitivas sobre la estructura de datos. Teniendo esto, cuando un usuario quiere generar la visualización, este registro de operaciones es serializado y enviado al *front-end*. Esto se logra utilizando la librería IPython Widgets en combinación con los serializadores y deserializadores definidos para el modelo.

El *front-end*, implementado en TypeScript, recibe el modelo serializado, lo deserializa y genera la visualización a partir del resultado. Para generar la visualización utiliza la librería D3js, que permite manipular el DOM (Document Object Model) en función de los datos del modelo.

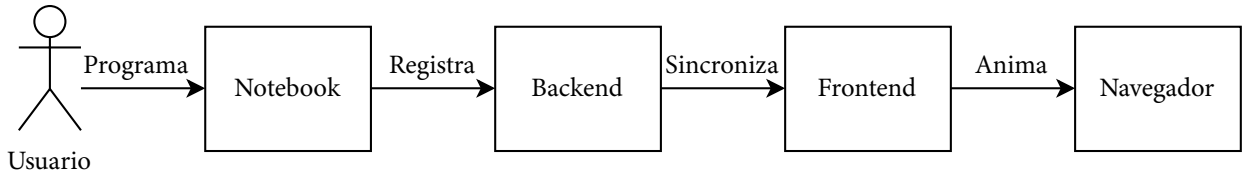


Figura 4.1: Flujo de la información

La figura 4.1 es un diagrama que muestra el flujo de la información cuando se utiliza la herramienta. Primero el usuario implementa y usa la estructura de datos en el notebook, después a partir de esto

se genera un registro en el back-end, luego este registro se sincroniza con el front-end y finalmente este último genera la animación en el navegador a partir del registro.

En la figura 4.2 se puede ver un diagrama que representa la arquitectura física de la solución. El usuario interactúa con la herramienta usando el navegador, donde corre el front-end. Este se comunica usando HTTP y Websockets con el servidor de Notebooks, que se encarga de interactuar con el archivo del Notebook y se comunica con el kernel usando ZeroMQ —una librería de comunicación asíncrona orientada a mensajes—. El kernel contiene el intérprete de Python y es donde corre el back-end de la herramienta.

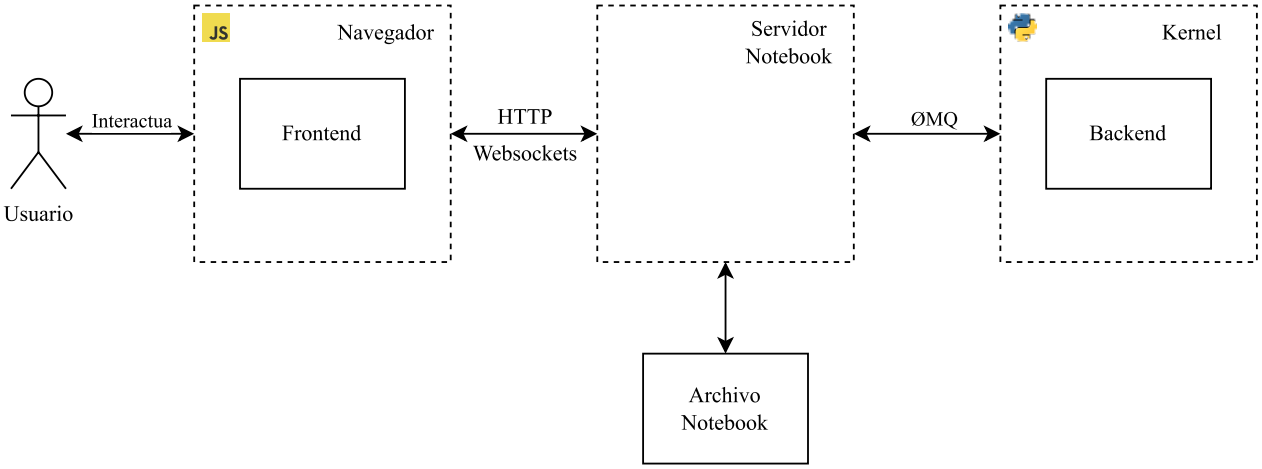


Figura 4.2: Diagrama de la arquitectura

4.2. Modelo de datos

Cuando un usuario quiere generar una visualización, el *back-end* le envía al *front-end* una representación de las operaciones realizadas sobre la estructura de datos. Para esto se diseñó un modelo que representa las operaciones primitivas que se pueden realizar sobre las listas enlazadas. Además, este modelo contiene metadatos que son útiles para generar la visualización.

El modelo que se utiliza para generar la visualización consiste en una lista de operaciones y metadatos de la *visualización*. Cada operación consiste en una operación primitiva y metadatos de la *operación*.

Las operaciones primitivas son las siguientes:

- Init(id, value, next) : Inicializar un nodo
- SetValue(id, value) : Asignar el valor de un nodo
- GetValue(id) : Obtener el valor de un nodo
- SetNext(id, next) : Asignar el siguiente de un nodo
- GetNext(id) : Obtener el siguiente de un nodo

donde *id* es el identificador único de cada nodo, *value* es una cadena de texto que representa el valor del nodo y *next* es el identificador del siguiente nodo en la lista enlazada o es *null*.

Los metadatos de la visualización son configuraciones de la visualización, por ejemplo, la duración de las transiciones y la duración de los fade-ins y fade-outs. En cambio, los metadatos de las operaciones

contienen información que solo es relevante para esa operación, por ejemplo, si se debe animar o no esa operación y el código fuente que dio origen a la operación.

En la figura 4.3 se puede ver un ejemplo de código fuente que usa la herramienta, una captura de la animación generada a partir de ese código y el modelo generado a partir del mismo código.

El modelo en realidad representa un grafo dirigido con un grado máximo de 1. Este nivel de flexibilidad es necesario porque como la representación se genera a partir de un programa escrito por el usuario, este puede contener errores que hagan que la estructura de datos implementada no sea necesariamente una lista enlazada. Por ejemplo, por ejemplo, el usuario puede crear nodos que no estén conectados entre sí y puede crear ciclos.

Para una versión futura de la herramienta se podría cambiar esta representación para que represente grafos sin un grado máximo. Esto permitiría representar estructuras de datos tales como grafos, árboles y listas doblemente enlazadas. Se consideró utilizar esta representación desde un principio, pero no se hizo porque dificultaba la generación de la visualización para la estructura de datos abordada.

4.3. Back-end

El back-end se encarga de proveer la instrumentación necesaria para que el usuario pueda generar visualizaciones de las estructuras de datos que ha implementado. Usando esta instrumentación mantiene un registro de las operaciones primitivas realizadas sobre la estructura de datos, y cuando el usuario quiere visualizar el resultado, serializa este registro y lo envía al *front-end*.

Para esto, la librería provee dos *decoradores* —azúcar sintáctica de Python para aplicar funciones a las definiciones de clases o funciones— *node* y *container*. El primero de estos se aplica a la clase que representa el nodo de la lista enlazada, mientras que el segundo se aplica a la clase que contiene la referencia al primer nodo de la lista. En el código 1 se puede ver un ejemplo del uso de estos decoradores y en la figura 4.4 se puede ver un cuadro de la visualización generada.

Para registrar las operaciones primitivas sobre la estructura de datos, el decorador *node* modifica los campos con los nombres pasados como parámetros, para que al momento de acceder o asignar a estos campos se registre la operación en el *logger*.

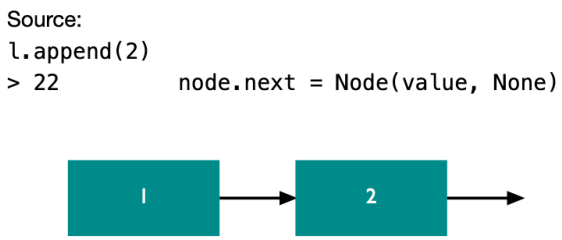
Para esto se implementaron las clases ValueField y NextField. Las instancias de estas clases, al ser utilizadas como atributos, registran en el *logger* sus accesos o asignaciones. Esto se hace utilizando los métodos `__get__` y `__set__`, métodos especiales de Python que se ejecutan al intentar acceder o asignar a objetos que tienen estos métodos definidos. El decorador *container* utiliza los nombres que recibe como parámetros para agregar como campos instancias de estas dos clases usando los nombres respectivos.

La clase Logger mantiene el registro de las operaciones primitivas sobre la estructura de datos. Además, se puede utilizar como un *context manager* (objetos de Python que definen un contexto y se utilizan con `with`) y dentro del *scope* o alcance introducido, todas las operaciones serán registradas en este *logger*. En el código 2 se puede ver un ejemplo de esta funcionalidad, la cual adicionalmente permite visualizar nodos que no están asociados a un contenedor.

El objeto *logger* implementa un método para registrar las distintas operaciones del modelo. Adicionalmente, este método se encarga de obtener el código fuente que dio origen a la operación,

```python
@node('value', 'next')
class Node:
    def __init__(self, value, next=None):
        self.value = value
        self.next = next

@container(lines_before=0, lines_after=0)
class List:
    def __init__(self):
        self.node = None

    def append(self, value):
        node = self.node
        if node is None:
            self.node = Node(value, None)
            return

        next = node.next
        while next is not None:
            node = next
            next = next.next
        node.next = Node(value, None)

l = List()
l.append(1)
l.append(2)
l.visualize()
```

(a) Código



(b) Captura de la visualización


```
{
  "operations": [
    {
      "operation": { "operation": "init", "id": 6, "value": "1", "next": null },
      "metadata": {
        "animate": true,
        "source": ["l.append(1)\n", "> 15     self.node = Node(value, None)\n"]
      }
    },
    {
      "operation": { "operation": "get_next", "id": 6 },
      "metadata": {
        "animate": true,
        "source": ["l.append(2)\n", "> 18 next = node.next\n"]
      }
    },
    {
      "operation": { "operation": "init", "id": 7, "value": "2", "next": null },
      "metadata": {
        "animate": true,
        "source": ["l.append(2)\n", "> 22 node.next = Node(value, None)\n"]
      }
    },
    {
      "operation": { "operation": "set_next", "id": 6, "next": 7 },
      "metadata": {
        "animate": true,
        "source": ["l.append(2)\n", "> 22 node.next = Node(value, None)\n"]
      }
    }
  ],
  "metadata": { "transition_duration": 1000, "fade_in_duration": 1000 }
}
```


(c) Modelo generado a partir del código

Figura 4.3: Código, captura de la visualización y modelo

```python
from dsvisualizer import node, container

@node('hd', 'tl')
class Node():
    def __init__(self, hd, tl):
        self.hd = hd
        self.tl = tl

@container()
class List():
    def __init__(self):
        self.head = None

    def push(self, v):
        self.head = Node(v, self.head)

l = List()
l.push(1)
l.push(2)
l.push(3)
l.visualize()
```

Código 1: Ejemplo de uso de la librería

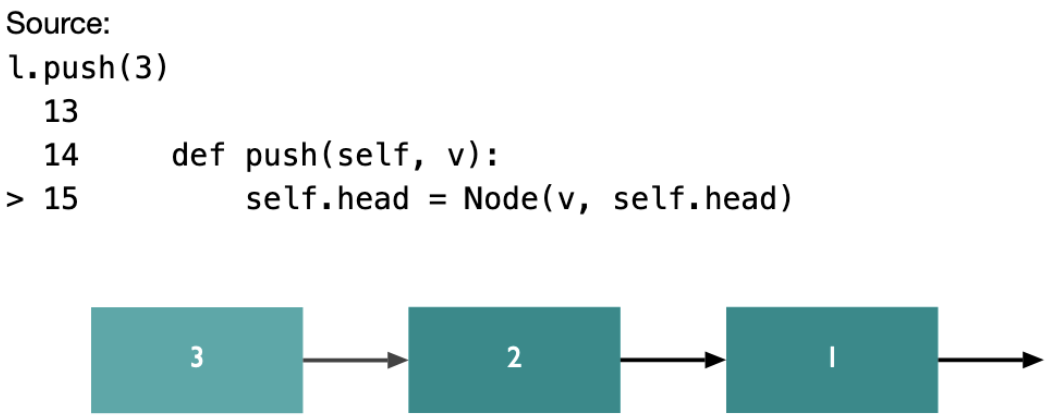


Figura 4.4: Captura de la visualización generada a partir del código 1.

para ser mostrado en la visualización. Para esto obtiene del *stack* de Python el número de la línea correspondiente y el nombre del archivo. Teniendo esto, lee una cantidad configurable de líneas de contexto del archivo y las formatea junto con el nombre y los argumentos del método del contenedor que dio origen a la operación.

Cuando se aplica el decorador *container* a una clase, además de agregar el método *visualize*, se crea un Logger asociado a cada instancia de esa clase, y para todos los métodos de esa clase se utiliza ese *logger* como un *context manager* para que las operaciones sobre la estructura de datos queden registradas en el *logger* del contenedor.

```
from dsvisualizer import logger

with Logger() as logger:
    n = Node(5, Node(10, Node(20, None)))
logger.visualize()

with logger:
    n = Node(10, n)
logger.visualize()
```

Código 2: Ejemplo de uso del *logger* como un *context manager*.

Esta parte originalmente se implementó usando herencia y metaclases en vez de decoradores, pero se cambió a la implementación actual utilizando decoradores porque de esta manera resulta más fácil la utilización de la herramienta. La versión que utilizaba herencia requería utilizar directamente las clases *ValueField* y *NextField*, además de tener que llamar explícitamente el constructor de la superclase en la clase que se está implementando. En cambio, esta versión solo requiere que el usuario agregue a su implementación los dos decoradores para poder usar la herramienta.

En el anexo A se puede ver la implementación de los decoradores *node* y *container*, además de la implementación de las clases *ValueField* y *NextField*.

4.4. Front-end

El front-end se encarga de generar la visualización a partir del modelo que se mantiene sincronizado con el back-end. Para esto, requiere saber cómo deserializar el modelo que recibe del back-end y generar la visualización a partir del resultado. Para generar la visualización utiliza la librería D3js.

Cuando el usuario llama el método *visualize* en el back-end esto genera un Widget de IPython que se sincroniza con el front-end. Este utiliza la función de deserialización para obtener un objeto de JavaScript a partir del modelo serializado. Después de esto, para dibujar el Widget, la librería utiliza la *vista* del Widget correspondiente. Esta obtiene un elemento del DOM —el output de la casilla que llamó el método *visualize*— y añade la visualización a ese elemento.

La vista del Widget primero inicializa la visualización y luego muestra paso por paso cada operación registrada. Al momento de mostrar la operación también se muestra el código fuente que dio origen a esa operación.

4.4.1. Visualizar una operación

Para visualizar una operación, primero se actualiza el modelo de la visualización y luego se animan los cambios. La animación de los cambios consiste en mostrar la iteración (cuando la operación es *GetNext*), con una flecha que se mueve de un nodo al siguiente, y luego actualizar la visualización según el nuevo modelo.

El modelo de la visualización está compuesto por un diccionario de nodos y un diccionario de arcos. El diccionario de nodos asocia los índices de los nodos a sus valores, mientras que el diccionario de arcos asocia los índices de los nodos al índice del nodo siguiente respectivo. Este modelo es apropiado

porque, como se explicó en la sección 4.2 el registro capturado describe una serie de operaciones sobre un grafo dirigido de grado máximo 1.

El modelo de la visualización se actualiza según el algoritmo 1. Dado un diccionario de nodos V, un diccionario de arcos E y una operación, se actualizan estos diccionarios cómo se describe en el pseudocódigo.

Algoritmo 1 Algoritmo para actualizar la representación de la visualización

```
1: function UPDATE(V, E, operation)
2:     if operation is Init(id, value, next) then
3:         INSERT(V, id, value)
4:         INSERT(E, id, next)
5:     else if operation is SetValue(id, value) then
6:         INSERT(V, id, value)
7:     else if operation is SetNext(id, next) then
8:         INSERT(E, id, next)
```

Para visualizar por separado las componentes conexas del grafo se deben encontrar los subgrafos conexos a partir de esta representación. En la literatura existen varios algoritmos para resolver este problema, pero por simplicidad se decidió acotar la librería al caso donde el grafo es acíclico, es decir, un bosque. De esta manera, todos los nodos que no tienen ningún arco que apunte a ellos son puntos de entrada. Además, todos estos puntos de entrada definen una lista conexa que no tiene conexiones con otras listas. Entonces, podemos visualizar cada una de estas componentes conexas como una lista. En el algoritmo 2 se puede ver el algoritmo utilizado para encontrar los puntos de entrada o *cabezas*.

Algoritmo 2 Algoritmo para encontrar las *cabezas*

```
1: function FINDHEADS(V, E)
2:     values ←SET(VALUES(E))
3:     heads ←FILTER(λn.n ∉ values, KEYS(V))   ▷ Índices de los nodos con grado de entrada 0
4:     return heads
```

Para generar la animación se visualiza en orden cada operación primitiva que se obtiene del *front-end* (animándola o no dependiendo de los metadatos de la operación). Para cada operación primitiva: Primero, se obtienen las listas conexas usando el algoritmo descrito previamente. Segundo, se genera una lista con los datos asociados a cada nodo que son necesarios para la visualización. Tercero, usando los datos de cada nodo, se anima primero la actualización de los nodos existentes y después la entrada de nuevos nodos. Esto se puede ver en el algoritmo 3

Algoritmo 3 Algoritmo para generar la animación

```
1: function DISPLAY(V, E)
2:     heads ←FINDHEADS(V, E)
3:     data ←DATA(V, E, heads)          ▷ Datos necesarios para dibujar cada nodo
4:     UPDATE(data)                     ▷ Muestra la actualización de los nodos existentes
5:     ENTER(data)                      ▷ Fade in de los nuevos nodos
6:     return
```

Los datos asociados a cada nodo son: el índice de la lista a la que pertenece, el largo de la lista a la que pertenece, su índice dentro de la lista a la que pertenece, su identificador único y su valor.

Teniendo la lista, usando D3js se asocian los nodos de la visualización a cada elemento en esta lista usando el identificador del elemento. La animación tiene dos fases: primero se anima la actualización de los nodos existentes y luego se anima la entrada de nuevos nodos.

En la primera fase se anima la transición a la nueva posición y valor del nodo, dados por los datos en el elemento de la lista correspondiente al nodo respectivo. En la segunda fase se hace un *fade in* de los nuevos nodos en sus posiciones correspondientes según los datos de la lista. Es importante el orden de estas fases, porque si se ejecutara en otro orden, los nodos entrantes podrían aparecer detrás de nodos preexistentes, dificultando la comprensión de la visualización.

4.5. Integración Continua y Entrega Continua

Para que la herramienta pueda ser utilizada instalándola con el administrador de paquetes de Python, se publicó en el repositorio oficial del administrador de paquetes de Python, PyPi (Python Package Index)[1]. Adicionalmente, cómo el front-end de la herramienta está implementado en TypeScript, también fue necesario publicar un paquete en el repositorio NPM (Node Package Manager)[2].

En el caso de Python, generar el paquete es relativamente sencillo, porque solo utiliza el código definido en el archivo `setup.py`. En cambio, en el caso de TypeScript, esto involucra el uso de una serie de herramientas. Algunas de estas herramientas son: el compilador de TypeScript (TSC) para el chequeo de tipos y la generación del código en JavaScript; y Webpack para generar *bundles* minimizadas a partir del código en JavaScript obtenido de TSC. La generación de *bundles* minimizadas es importante porque reduce el tamaño del paquete que tiene que instalar el usuario, lo que es especialmente relevante cuando este no cuenta con una buena conexión a internet, y porque reduce el tiempo inicial que el navegador se demora en poder ejecutar el código.

Para facilitar el lanzamiento de nuevas versiones de la herramienta, el repositorio se encuentra en GitHub y utiliza Github Actions —una funcionalidad de GitHub que permite ejecutar programas después de eventos como commits o deploys— para correr las pruebas y para publicar versiones nuevas de la librería.

Cada vez que se agrega un commit al repositorio, una GitHub Action corre los tests y los *linters* de Python (Black) y de JavaScript (Prettier). Los tests se corren tanto en Ubuntu como en macOS y para Python versión 3.7 y 3.10. Además, cuando se hace un release del repositorio, otra Github Action compila los paquetes de Python y de JavaScript y los sube a los repositorios de los administradores de paquetes respectivos. Esto permite lanzar nuevas versiones de la herramienta de forma muy expedita, facilitando y acelerando el desarrollo.

Gracias a que la librería se encuentra publicada en los repositorios respectivos puede ser utilizada por cualquier usuario con acceso a estos repositorios. Además, se comprobó que la herramienta funciona no solo en Jupyter Notebook, sino también en Google Colab y en Visual Studio Code.

[1] El paquete de PyPi se encuentra disponible en `https://pypi.org/project/dsvisualizer/`

[2] El paquete de NPM se encuentra disponible en `https://www.npmjs.com/package/jupyter-dsvisualizer`

Capítulo 5

Validación

Para evaluar la solución se hicieron pruebas con usuarios y se les pidió que contestaran un cuestionario. Los participantes fueron estudiantes de la Facultad de Ciencias Físicas y Matemáticas que accedieron a ser parte de la investigación. Estos fueron reclutados usando el foro institucional, el foro del curso CC3001 y el grupo de Telegram de los estudiantes del Departamento de Ciencias de la Computación.

Para realizar estas pruebas se le pidió la autorización al comité de ética y todos los usuarios firmaron un consentimiento informado antes de participar. En el anexo C se puede ver la certificación del Comité de Ética y Bioseguridad para la Investigación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Adicionalmente, en el anexo D se puede ver el consentimiento informado que fue aprobado por el comité y fue firmado por los participantes.

Las pruebas con usuarios consistieron en que, a cada participante, primero se le explicó el proceso, luego se le dio una breve introducción sobre cómo usar la herramienta, después se le pidió que utilice la herramienta y, finalmente se le pidió que conteste un cuestionario.

El cuestionario tiene tres partes. La primera parte consiste en el System Usability Scale (SUS) —un método estándar para evaluar la usabilidad de sistemas informáticos—, la segunda parte corresponde a preguntas abiertas sobre la herramienta, y la tercera parte corresponde a una caracterización del participante. En el anexo B se puede ver el cuestionario que fue utilizado.

La escala SUS fue introducida en [2] como un método sencillo para evaluar la usabilidad de sistemas informáticos y se ha vuelto un método estándar en la industria para medir la usabilidad de todo tipo de sistemas, en la sección 2.4 se describe con más detalle esta escala. En [1] analizan 10 años de datos de SUS y encuentran que esta escala es una herramienta altamente robusta y versátil para las evaluaciones de usabilidad. Como la versión original está en inglés, se utilizó la versión en español adaptada y validada en [11], con una modificación. Esta consistió en que se cambió "*Me sentí muy confiado al usar la herramienta*" por "*Me sentí muy confiado o confiada al usar la herramienta*" con el objetivo de hacer el cuestionario más inclusivo.

Las preguntas abiertas que se le hicieron los participantes fueron: "*¿Qué te gustó de la herramienta? ¿Por qué?*", "*¿Qué no te gustó de la herramienta? ¿Por qué?*", y "*¿Cómo podría ser mejor la herramienta?*" Se escogieron estas preguntas para tener una idea de porque a los usuarios les gustó o no la herramienta y también de cómo esta se podría mejorar en el futuro.

En la caracterización se preguntó por edad, género y avance curricular. La edad se preguntó en

intervalos de 2 años entre los 18 años y 30 o más años. Se preguntó de esta manera para evitar poder identificar a los participantes a partir de sus respuestas. Para el género las opciones eran *hombre*, *mujer*, *otro* y *prefiero no responder*. Para el avance curricular se les pidió a los estudiantes que marcaran todos los cursos que han cursado o estaban cursando de una lista de 5 cursos obligatorios de Ingeniería Civil en Computación. Cómo los cursos seleccionados requieren tomar el curso anterior en la lista, esta respuesta se puede representar con un número del 0 al 5, donde el número representa que el estudiante ha tomado todos los cursos de la lista hasta ese número.

Se preguntó por edad y género porque en [1] se observa cierta relación entre estos factores y el puntaje obtenido en SUS. Se preguntó por avance curricular porque se cree que puede haber una relación entre el nivel de conocimiento en programación o en ciencias de la computación y la facilidad para usar la herramienta.

En total participaron 12 estudiantes. De estos, 10 se identificaron como hombres, 1 como mujer y 1 como otro. La edad promedio de los participantes fue 22 años[1]. En cuanto al avance curricular, 9 de los estudiantes marcaron 2, lo que significa que han cursado los cursos de la lista hasta Algoritmos y Estructuras de Datos. En la figura 5.1 se puede ver la distribución de género y en la tabla 5.1 se puede ver el promedio y desviación estándar de la edad y avance curricular.

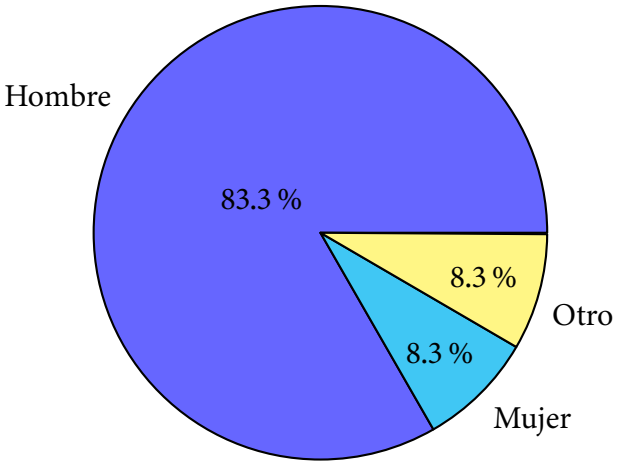


Figura 5.1: Distribución de género

Las respuestas de los participantes se pueden ver en la tabla 5.2. Se muestra el avance, edad, el puntaje por pregunta para cada pregunta y el puntaje total, la última fila muestra el promedio para cada columna. Cómo el cuestionario SUS alterna afirmaciones positivas con negativas, para facilitar la compresión de los resultados se presentan las respuestas, de forma que los puntajes están en una escala entre 0 y 4, donde 0 es lo peor y 4 es lo mejor. Utilizando el sistema de puntuación de SUS se obtuvo un puntaje promedio de 90.0, con una desviación estándar de 7.6.

Para interpretar este puntaje es preferible calcular el percentil en el que quedaría este puntaje dentro de un grupo grande de evaluaciones realizadas utilizando la escala SUS. Por esto, se utilizó la tabla de [10] para transformar el puntaje a un percentil, obteniendo que el puntaje se encuentra en el percentil 99.8.

En cuanto a las preguntas abiertas. En la primera pregunta, *¿Qué te gustó de la herramienta? ¿Por qué?*, las respuestas más frecuentes mencionaban que la herramienta muestra el código fuente que

[1]El promedio de las edades se calculó utilizando la marca de clase de los intervalos correspondientes

Tabla 5.1: Resumen de los resultados

	Promedio	**Desviación**
Puntaje	90.0	7.6
Avance curricular	3.3	1.2
Edad	22	1.5

Tabla 5.2: Matriz de respuestas, transformando los puntajes de los ítems a una escala entre 0 y 4 donde mayor es mejor.

Avance	**Edad**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**Puntaje**
2	20–21	4	4	3	4	4	4	3	4	3	2	87.5
2	22–23	4	4	4	3	4	4	4	4	4	3	95.0
5	24–25	1	4	4	4	2	4	4	4	3	1	77.5
4	22–23	1	4	4	1	4	4	4	4	2	3	77.5
2	20–21	3	4	4	4	4	4	4	4	3	1	87.5
2	22–23	4	4	3	3	3	4	3	3	3	4	85.0
2	22–23	1	4	4	4	4	4	4	4	4	4	92.5
2	22–23	4	4	4	4	4	4	4	4	4	4	100.0
2	20–21	4	4	4	3	3	4	4	3	3	4	90.0
2	20–21	4	4	4	4	4	4	4	4	4	4	100.0
2	20–21	3	4	4	4	4	4	4	4	4	4	97.5
5	24–25	4	3	4	4	4	4	3	4	3	3	90.0
2.7	22.0	3.1	3.9	3.8	3.5	3.7	4.0	3.8	3.8	3.3	3.1	90.0

dio origen a la operación y que solo requiere agregar dos líneas de código para usarla. En la segunda pregunta, *¿Qué no te gustó de la herramienta? ¿Por qué?*, no hubo respuestas significativamente más comunes que otras. Sin embargo, una respuesta que se repitió dos veces tiene que ver con que al eliminar un nodo de la lista este visualmente se mueve hacia debajo la lista original en vez de desaparecer. En la tercera pregunta, *¿Cómo podría ser mejor la herramienta?*, la respuesta más frecuenta estaba relacionada permitir visualizar otras estructuras de datos que no sean listas enlazadas. En el anexo E se pueden ver todas las respuestas a las preguntas abiertas.

Estos resultados muestran que la herramienta soluciona el problema abordado y cumple con el requisito de permitirle al usuario utilizar la herramienta agregando la menor cantidad de instrumentación necesaria a su código. Sin embargo, también presenta oportunidades de mejora. La principal oportunidad de mejora tiene que ver con permitir generar visualizaciones de otras estructuras de datos. Otra oportunidad de mejora sería mostrar en la visualización las referencias a los nodos del *contenedor*. Esto ayudaría a los usuarios a entender qué está pasando en algunos de los casos que ocurrieron durante las pruebas con usuarios. Algunas oportunidades de mejora de menor importancia tienen que ver con hacer la herramienta más configurable, permitiendo cambiar variables como: color de fondo, color de los nodos, forma de los nodos, tamaño de los nodos, etc.

Capítulo 6

Conclusión

Se desarrolló *dsvisualizer*, una herramienta para generar visualizaciones animadas de estructuras de datos para ser utilizada en Jupyter Notebooks. Esta herramienta permite a los usuarios visualizar sus propias implementaciones, agregando la menor cantidad de instrumentación necesaria a su código. Además, se evaluó esta herramienta realizando pruebas con 12 estudiantes de la Facultad de Ciencias Físicas y Matemáticas. El resultado de esta evaluación fue muy positivo, obteniendo un puntaje promedio de 90.0, el cual se encuentra en el percentil 99.8.

Se cumplió el objetivo general de "*Diseñar e implementar una librería para generar visualizaciones de estructuras de datos en Notebooks de Python que sea efectiva, eficiente y usable*" y también se cumplieron todos los objetivos específicos.

Esta herramienta podrá ser utilizada cómo una ayuda para la docencia en el curso Algoritmos y Estructuras de Datos de la Facultad de Ciencias Física y Matemáticas, que beneficiará a los futuros estudiantes de esta facultad. Además, se encuentra disponible públicamente, por lo que cualquier persona que le interese podrá utilizarla para ayudarla en su entendimiento de las estructuras de datos. La herramienta cuenta con licencia BSD (Berkeley Software Distribution), lo que permite redistribuirla, usarla y modificarla con mínimas restricciones.

Durante el desarrollo de la memoria se aprendió mucho sobre distintos temas. Se aprendió sobre la generación de visualizaciones, primero utilizando Threejs y luego utilizando D3js al descubrir que Threejs no era la herramienta apropiada para este problema. Se aprendió sobre distintos mecanismos del lenguaje Python, la primera versión de la librería utilizaba herencia, luego se pasó por una versión que utilizaba metaclases y finalmente, se optó por utilizar decoradores, ya que esto permite una interfaz más cómoda para el usuario. Se aprendió sobre metodologías para la evaluación de usabilidad y sobre cómo aplicarlas.

A partir del trabajo desarrollado sería muy interesante extenderlo para permitir visualizar otras estructuras de datos, fue la mejora más pedida durante las pruebas con usuarios. La herramienta define una base en cuanto a la manera de capturar las operaciones realizadas por el usuario, al modelo de datos y a la visualización que podría ser expandida para otras estructuras de datos. Sería particularmente interesante extender la herramienta a árboles, ya que junto con las listas enlazadas es la estructura de datos que más se estudia en los cursos básicos de Ciencias de la Computación. Otras estructuras de datos que también sería interesante agregar son: grafos, arreglos, listas circulares, tablas de hash y listas doblemente enlazadas.

Otro trabajo futuro que se podría realizar es aprovechar la representación intermedia que utiliza la herramienta para crear más front-ends o más back-ends. Por ejemplo, utilizando el modelo actual se podrían implementar front-ends que generen visualizaciones utilizando GraphViz y también se podría implementar back-ends en otros lenguajes, cómo por ejemplo JavaScript.

Bibliografía

[1] Aaron Bangor, Philip T. Kortum y James T. Miller. «An Empirical Evaluation of the System Usability Scale». En: *International Journal of Human–Computer Interaction* 24.6 (2008), págs. 574-594. DOI: 10.1080/10447310802205776. eprint: https://doi.org/10.1080/10447310802205776. URL: https://doi.org/10.1080/10447310802205776.

[2] John Brooke. «SUS: A 'Quick' and 'Dirty' Usability Scale». En: *Usability Evaluation in Industry*. Ed. por Patrick W. Jordan y col. Taylor y Francis, jun. de 1996. Cap. 21, págs. 189-194. ISBN: 9780748404605.

[3] Thomas H. Cormen y col. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[4] GitHub. *The State of the Octoverse*. 2021. URL: https://octoverse.github.com/#top-languages-over-the-years.

[5] Christopher D. Hundhausen, Sarah A. Douglas y John T. Stasko. «A Meta-Study of Algorithm Visualization Effectiveness». En: *Journal of Visual Languages and Computing* 13.3 (2002), págs. 259-290. DOI: 10.1006/jvlc.2002.0237. URL: https://doi.org/10.1006/jvlc.2002.0237.

[6] Jupyter Project. *Architecture*. 2021. URL: https://jupyter.readthedocs.io/en/latest/projects/architecture/content-architecture.html.

[7] Jupyter Widgets. *User Guide*. 2021. URL: https://ipywidgets.readthedocs.io/en/stable/user_guide.html.

[8] Thomas Kluyver y col. *Jupyter Notebooks-a publishing format for reproducible computational workflows*. Vol. 2016. 2016.

[9] Terence Parr. *lolviz*. 2021. URL: https://github.com/parrt/lolviz.

[10] Jeff Sauro y James R. Lewis. *Quantifying the User Experience: Practical Statistics for User Research*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123849687.

[11] Magdalena Del Rocio Sevilla-Gonzalez y col. «Spanish Version of the System Usability Scale for the Assessment of Electronic Tools: Development and Validation». En: *JMIR Hum Factors* 7.4 (dic. de 2020), e21161. ISSN: 2292-9495. DOI: 10.2196/21161. URL: http://www.ncbi.nlm.nih.gov/pubmed/33325828.

[12] Ivan Sipirán. *AED - Utilities*. 2021. URL: https://github.com/ivansipiran/aed-utilities.

[13] Nick Stanch. *Reftree*. 2021. URL: https://github.com/stanch/reftree.

[14] The Manim Community Developers. *Manim – Mathematical Animation Framework*. Ver. v0.16.0. Jul. de 2022. URL: https://www.manim.community/.

[15] Katherine Ye y col. «Penrose: From Mathematical Notation to Beautiful Diagrams». En: *ACM Transactions on Graphics* 39.4 (jul. de 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392375. URL: https://doi.org/10.1145/3386569.3392375.

Anexos

Anexo A

Código fuente de los decoradores

```
from functools import wraps
import itertools
from types import FunctionType

from dsvisualizer.operations import Init, GetNext, GetValue, SetNext, SetValue
from dsvisualizer.logger import Logger, get_logger

counter = itertools.count()

class Uninitialized:
    def __repr__(self) -> str:
        return "uninitialized"

UNINITIALIZED = Uninitialized()

class ValueField:
    def __set_name__(self, owner, name):
        self.name = name

    def __get__(self, obj, objtype=None):
        get_logger().log(GetValue(obj._id))
        return obj._value

    def __set__(self, obj, value):
        if obj._value != UNINITIALIZED:
            get_logger().log(SetValue(obj._id, str(value)))
        obj._value = value

class NextField:
    def __set_name__(self, owner, name):
        self.name = name

```

```
    def __get__(self, obj, objtype=None):
        get_logger().log(GetNext(obj._id))
        return obj._next

    def __set__(self, obj, next):
        if obj._next != UNINITIALIZED:
            get_logger().log(SetNext(obj._id, next._id if next else None))
        obj._next = next

def wrapper(method):
    @wraps(method)
    def wrapped(*args, **kwargs):
        with args[0]._logger:
            res = method(*args, **kwargs)
        return res

    return wrapped

def container(lines_before=2, lines_after=2):
    def decorator(cls):
        init = cls.__init__

        def __init__(self):
            self._logger = Logger(lines_before=lines_before, lines_after=lines_after)
            init(self)

        def visualize(self, transition_duration=1000, fade_in_duration=1000):
            return self._logger.visualize(
                transition_duration=transition_duration,
                fade_in_duration=fade_in_duration,
            )

        for name in dir(cls):
            value = getattr(cls, name)
            if isinstance(value, FunctionType) and name != "__init__":
                setattr(cls, name, wrapper(value))

        setattr(cls, "__init__", __init__)
        setattr(cls, "visualize", visualize)
        return cls

    return decorator

def node(value_field: str = "value", next_field: str = "next"):
    def decorator(cls):
        init = cls.__init__

```

```
        setattr(cls, value_field, ValueField())
        setattr(cls, next_field, NextField())

        def __init__(self, *args, **kwargs):
            self._next = UNINITIALIZED
            self._value = UNINITIALIZED
            self._id = next(counter)
            init(self, *args, **kwargs)

            if value_field in kwargs:
                value = kwargs[value_field]
                n = kwargs.get(next_field, None)._id
            else:
                value = args[0]
                n = args[1]._id if args[1] else None

            get_logger().log(Init(self._id, str(value), n))

        def __repr__(self):
            return f"({self._get_class_name()} {self._value} {self._next})"

        def _get_class_name(self):
            return self.__class__.__name__

        def visualize(self, transition_duration=1000, fade_in_duration=1000):
            return get_logger().visualize(
                transition_duration=transition_duration,
                fade_in_duration=fade_in_duration,
            )

        setattr(cls, "__init__", __init__)
        setattr(cls, "__repr__", __repr__)
        setattr(cls, "_get_class_name", _get_class_name)
        setattr(cls, "visualize", visualize)

        return cls

    return decorator
```

Anexo B

Cuestionario

Escala de Usabilidad de Sistemas (SUS)

Por favor seleccione de cada uno de los enunciados la opción que mejor describa su experiencia con la herramienta. Un puntaje de 1 significa que usted se encuentra totalmente en desacuerdo con el enunciado, mientras que un puntaje en 5 significa que está totalmente de acuerdo, un puntaje de 3 significaría que usted se encuentra neutral con el enunciado.

	Totalmente en desacuerdo				**Totalmente de acuerdo**
1. Me gustaría usar esta herramienta frecuentemente	1	2	3	4	5
2. Considero que esta herramienta es innecesariamente compleja	1	2	3	4	5
3. Considero que la herramienta es fácil de usar	1	2	3	4	5
4. Considero necesario el apoyo de personal experto para poder utilizar la herramienta	1	2	3	4	5
5. Considero que las funciones de la herramienta están bien integradas	1	2	3	4	5
6. Considero que la herramienta presenta muchas contradicciones	1	2	3	4	5
7. Imagino que la mayoría de las personas aprenderían a usar esta herramienta rápidamente	1	2	3	4	5
8. Considero que el uso de esta herramienta es tedioso	1	2	3	4	5
9. Me sentí muy confiado al usar la herramienta	1	2	3	4	5
10. Necesité saber bastantes cosas antes de poder empezar a usar la herramienta	1	2	3	4	5

Preguntas abiertas

¿Qué te gustó de la herramienta? ¿Por qué?

¿Qué no te gustó de la herramienta? ¿Por qué?

¿Cómo podría ser mejor la herramienta?

Caracterización

1. Indica tu edad
    - ☐ 18–19
    - ☐ 20–21
    - ☐ 22–23
    - ☐ 24–25
    - ☐ 26–27
    - ☐ 28–29
    - ☐ 30 o más
2. Indica tu género
    - ☐ Masculino
    - ☐ Femenino
    - ☐ Otro,
    - ☐ Prefiero no responder
3. Marca los cursos que hayas tomado o estés tomando
    - ☐ Introducción a la programación
    - ☐ Algoritmos y estructuras de Datos
    - ☐ Diseño y análisis de algoritmos
    - ☐ Lenguajes de programación
    - ☐ Proyecto de software

Anexo C

Certificación Comité de Ética y Bioseguridad

CERTIFICACIÓN Nº 050
COMITÉ DE ÉTICA Y BIOSEGURIDAD PARA LA INVESTIGACIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

El Comité de Ética y Bioseguridad para la Investigación de la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile certifica haber analizado el proyecto titulado **"Librería de Visualización de Estructuras de Datos"** cuyo jefe de proyecto es el profesor **Iván Sipirán Mendoza**, del Departamento de Ciencias de la Computación, FCFM, Universidad de Chile.

Los participantes serán estudiantes de la Facultad de Ciencias Físicas y Matemáticas y serán reclutados usando el foro institucional y el grupo de Telegram de los estudiantes del Departamento de Ciencias de la Computación. Los participantes firmarán un Consentimiento Informado y deberán llenar un formulario sobre las actividades con la librería de visualización de datos. Los datos obtenidos serán almacenados en los computadores de los investigadores separados de la información identificable de los usuarios.

La metodología y los documentos de Consentimiento Informado permiten certificar que:

i) El proyecto cumple con los estándares nacionales e internacionales de ética de la investigación, de acuerdo a la Declaración Universal de los Derechos Humanos, el Pacto de Derechos Civiles y Políticos, el Pacto de Derecho Económicos Sociales y Culturales, las leyes chilenas y el Documento oficial de ética para la investigación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.
ii) El Comité de Ética considera que la investigación no vulnera la dignidad de los sujetos, no constituye una amenaza bajo ninguna circunstancia ni causa daño.

iii) Asimismo, el Comité que suscribe auditará, al termino del proyecto, el cumplimiento de los estándares arriba enunciados propios de la disciplina involucrada y la correcta aplicación del modelo de consentimiento informado para asegurarse del cumplimiento. Esta auditoría, además, tiene el propósito de asegurar la garantía del derecho a la privacidad, confidencialidad y el anonimato de los sujetos involucrados en la investigación.

iv) Dejamos constancia que el profesor Sipirán será responsable por eventuales daños causado a las personas por errores que puedan cometerse durante la investigación.

Dr. Carlos Conca R.
Profesor Titular
Departamento de Ingeniería Matemática

Firmado digitalmente por
Dra. Marcela Munizaga M.
Profesora Titular
Directora Académica y de Investigación

Dr. Manuel Patricio Jorquera E.
Secretario

Comité de Ética y de Bioseguridad para la Investigación
UNIVERSIDAD DE CHILE
Facultad de Ciencias Físicas y Matemáticas

Santiago, 05 de julio de 2022

MMM/CCR/PJE/rox.

Anexo D

Consentimiento informado

CONSENTIMIENTO INFORMADO

Le invitamos a usted, en su calidad de *estudiante,* a colaborar con el proyecto de Investigación **"Librería de Visualización de Estructuras de Datos",** de la *Escuela de Ingeniería o de la FCFM* de la Universidad de Chile. A continuación, le proporcionamos información detallada del proyecto y de los términos concretos de su participación, con el fin de ayudarlo(a) a decidir si desea o no colaborar con el mismo.

Información sobre el proyecto

1. Este Proyecto está a cargo del Profesor Ivan Sipirán, académico de la Universidad de Chile.
2. El objetivo general de este proyecto es evaluar la usabilidad de una herramienta de visualización de estructuras de datos.
3. Durante la investigación se recogerá información en diversos formatos tales como cuestionarios.
4. El manejo de los datos tendrá carácter estrictamente confidencial. La información recogida se usará de manera exclusiva para fines asociados a la presente investigación.
5. Los datos recolectados dentro del Proyecto, serán administrados resguardando la estricta confidencialidad de la identidad de las personas participantes. Sólo los investigadores tendrán acceso a los datos desagregados, y los datos identificables, tales como los corres electrónicos de los participantes, serán eliminados después de la investigación.
6. Los datos recolectados dentro del Proyecto serán usados exclusivamente para ser publicados en revistas científicas, memorias, libros especializados y en congresos académicos en Chile y en el extranjero y, en ningún caso estas publicaciones contendrán información que permita identificar a los participantes de la investigación o información que pudiese conducir a su identificación. Este estudio ha sido aprobado por el Comité de Ética y Bioseguridad para la Investigación, de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

Información sobre la participación de los(as) estudiantes en el proyecto.

1. Su participación consistirá en:
   Utilizar la herramienta y luego contestar un cuestionario sobre su experiencia con la herramienta. Esto debería tomar alrededor de 15 minutos.
2. La recolección de los datos anteriores se realizará mediante un cuestionario en papel en las dependencias de la Facultad de Ciencias Físicas y Matemáticas o mediante una sesión zoom de manera que no implica ningún tipo de desplazamiento adicional.

3. La participación de los(as) *estudiantes* en el proyecto no contempla ningún tipo de remuneración económica ni retribución material de ningún tipo, ni por parte del investigador responsable ni de las instituciones participantes.
4. No existen ni riesgos ni beneficios o compensaciones específicas asociadas a la colaboración del estudiante con el proyecto.

Derechos de los(as) estudiantes que colaboran con el proyecto

1. Su participación en este proyecto es total y absolutamente voluntaria.
2. Usted puede revocar la decisión de participar en una parte o en la totalidad del Proyecto en cualquier momento, sin perjuicio alguno y sin tener que explicar o justificar su decisión.
3. Usted conocerá los resultados finales de la investigación si lo desea. Para ello el investigador responsable le preguntará si desea conocer los resultados finales y en el caso de que si le interese le pedirá su correo electrónico y se le enviará un documento que explicará los resultados.
4. Frente a cualquier inquietud relativa a este estudio, usted podrá contactarse con el investigador responsable a través del correo electrónico, a la dirección jose.romero@ing.uchile.cl
5. Otras preguntas pueden realizarse a la Presidenta del Comité de Ética y Bioseguridad para la Investigación, de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, **Prof. Marcela Munizaga**, a quien usted podrá contactar a sediraca@ing.uchile.cl.
6. Debido a lo anterior, le rogamos indicar su aceptación a participar en el estudio y a que se publiquen sus resultados, los que no incluirán su nombre. Deberá firmar dos copias de los documentos, una quedará en su poder.

CONSENTIMIENTO INFORMADO – ESTUDIANTE

PARTE 2: FORMULARIO DE CONSENTIMIENTO

Declaración del (la) Estudiante:

He sido invitado/a a colaborar con el proyecto de Investigación **"Librería de Visualización de Estructuras de Datos",** de la Escuela de Ingeniería y FCFM de la Universidad de Chile. He leído (o se me ha leído) la información del documento de consentimiento. He tenido tiempo para hacer preguntas y se me ha contestado claramente. No tengo ninguna duda sobre mi participación. Acepto voluntariamente participar y sé que tengo el derecho a terminar mi participación en cualquier momento sin indicar causas y sin consecuencias negativas para mí. Acepto además que se usen cuestionarios para registrar mi participación en la investigación.

De acuerdo a las condiciones antes descritas, declaro que [marcar con una X]:

	No acepto utilizar la herramienta y luego contestar un cuestionario, lo que debería tomar alrededor de 15 minutos.
	Sí acepto utilizar la herramienta y luego contestar un cuestionario, lo que debería tomar alrededor de 15 minutos.

Nombre del (la) estudiante:
RUT del (la) estudiante:

_____
[Firma del (la) estudiante]

José Luis Romero Munizaga investigador responsable del proyecto de investigación identificado, suscribe el compromiso de respetar cabalmente las condiciones detalladas.

_____
[Firma del investigador responsable]

En Santiago a…….....de 20__.

(Se firman dos copias, quedando una en poder de cada uno de los firmantes: el académico participante y el investigador principal del proyecto)

Consentimiento Informado Autorizado digitalmente por Marcela Munizaga M.

Anexo E

Respuestas

E.1. ¿Qué te gustó de la herramienta? ¿Por qué?

1. Sí, fue entretenido ver como se movía cada parte de la lista y me permitió ver más fácilmente los errores
2. Me gustó lo cómoda y fácil de usar la herramienta. No es necesario tanto conocimiento experto para usarla, y la visualización es muy intuitiva
3. Los colores, son estéticamente agradables
4. Que es bastante simple agregar a código que ya está escrito
5. Logra mostrar claramente y paso a paso funciones que uno está creando. Esto ayuda enormemente a entender gráficamente lo que se está haciendo
6. Pensando desde el punto de vista docente, me gustó porque me recordó la estructura de datos como la vi en cátedra por primera vez, las animaciones pueden ser aceleradas, lo cual me gustó. Además, el proceso que sigue la animación es bien claro y le hace completa justicia a la estructura de datos.
7. Me gustó que fuera recorriendo el código y mostrando la visualización. Me gustó el tamaño y el color.
8. Me gustó mucho que la visualización fuera mostrando la línea de código en la que estaba leyendo, pues ayuda enormemente a hallar errores en el código.
9. Me gustó la idea de poder trabajar en tiempo real viendo como se construye la lista enlazada, más aún con la recursión que puede ser compleja de visualizar a primera.
10. Me gustó que implementarlo requiera tan solo 2 líneas, es muy rápido.
11. Ayuda a entender la construcción de una estructura paso a paso.
12. Me gustó que fuera una visualización dinámica, es decir, muestra el proceso de los métodos llamados en la estructura de datos. Esto, porque me parece muy útil para entender lo que está sucediendo en la misma estructura.

E.2. ¿Qué no te gustó de la herramienta? ¿Por qué?

1. Creo que para listas largas el tiempo en que va mostrando la lista se hace muy largo
2. Nada, me gustó mucho
3. No parece haber forma de quitar nodos
4. Que no conocí de antemano las limitaciones, por lo tanto, algo que programé no servía (porque el nodo tenía dos çabezas")
5. No hay nada que realmente no me haya gustado. Lo único que podría mencionar es que la implementación de la herramienta se me hizo extraña, pero con la miniexplicación que se me presentó no tuve problema alguno.
6. La visualización es buena, pero si le agrego algo de más de 6 elementos en la lista enlazada se pierde la visualización de algunas casillas. Por otro lado, si le paso listas un poco grandes como elemento a la lista, estas se representan algo más grandes que la casilla que le corresponde, aunque esto último no es realmente una molestia, sino más bien una observación.
7. No poder ajustar la velocidad y apariencia, o que no tuviera el formato del apunte.
8. No le encontré ningún aspecto negativo
9.
10. El uso de decoradores hace que sea un poco extraño para estudiantes del curso de algoritmos, pues no los he usado antes en este curso o en cursos anteriores.
11. Es necesario que alguien explica que indica el puntero que va apuntando hacia abajo.
12. No me gustó que al principio no entendí por qué quedaban más nodos de los que debe haber en la lista, al eliminar nodos.

E.3. ¿Cómo podría ser mejor la herramienta?

1. Sería bueno añadir un parámetro de rapidez de visualización o algo de ese estilo
2. Quizás no poner los paréntesis en el decorador çontainerz que admita múltiples objetos en la visualización.
3. Implementando la eliminación de nodos
4. Listas muy largas no se muestran completas
5. Si se logra expandir y que no solo funcione para listas enlazadas.
6. Creo que solucionar de alguna manera la visualización para listas enlazadas para hasta 10 elementos sería bueno para la herramienta, pues pensando en la docencia al poder visualizar solo un par de datos es limitado.
7. Personalización: Color, tamaño, velocidad, fuente, figura. En su defecto, predeterminado sea parecido al apunte.

   Muestras o ejemplos: visualizar por defecto dos o más posibilidades.

Árboles: Suele ser la parte más difícil de entender, poder jugar con esto, serviría mucho más, pues las primeras listas se ven poco (poco relevante en comparación).

8. Expandirse a más estructuras de datos o incluir etiquetas en la visualización
9. Solamente aumentar sus usos, entendiendo que es la primera versión, le veo mucho potencial. Sentí que me faltó ver más usos, pero que fue para ver primeras impresiones.
10. Que haya una opción para que no muestre todo el recorrido del código a la hora de agregar un elemento a la lista, que solo se agregue al final.
11. Limitar que la figura se pueda salir del cuadro. Al tomarlo clickeando se puede sacar de lo visible de la pantalla.
12. Cambiando el color de los nodos que ya no forman parte de la estructura principal, o sea, los que quedan libres. Por ejemplo, en el caso de ser eliminados de la lista.