UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

ENTORNO GENÉRICO PARA LA VISUALIZACIÓN DEL FUNCIONAMIENTO DE LOS ALGORITMOS

Carlos Caride Santeiro

Dirigido por: Fernando López Ostenero

Curso: 2020-2021: Convocatoria septiembre

ENTORNO GENÉRICO PARA LA VISUALIZACIÓN DEL FUNCIONAMIENTO DE LOS ALGORITMOS

Proyecto de Fin de Grado en Ingeniería Informática de modalidad específica

Realizado por: Carlos Caride Santeiro

Dirigido por: Fernando López Ostenero

Tribunal calificador

Presidente: D/Dª.

Secretario: D/Dª.

Vocal: D/Dª.

Fecha de lectura y defensa: 16 de septiembre de 2021

Calificación:

Agradecimientos

Tengo que dar las gracias a toda la gente que me apoyó no sólo en los estudios de ingeniería informática, sino en toda mi vida.

Este camino lo empezaste tú, mamá. Aun desconociendo por completo que es la informática, veías en mí la capacidad de hacerlo y me empujaste a ello. A ti Raquel, por ser esa prima que se alió con mi madre para obligarme a realizar el grado.

A Laura, mi compañera de vida. ¿Qué quieres que te diga? Me has ayudado en momentos en los que los ánimos estaban bajos, has intentado que desconecte y pueda seguir, me has empujado cuando no tenía fuerzas. Para finalizar contigo, te lanzo una pregunta… ¿Sabes una cosa?

A todos aquellos que no he nombrado. En menor o mayor medida habéis ayudado a que esté donde estoy. Daros las gracias y seguid en mi vida, os necesito.

Y sobre todo a ti, papá. Aunque ya no estés con nosotros me has enseñado a que los problemas no te deben detener y que es mejor pararse un minuto a pensar antes de actuar. Me dejabas caer (siempre con tus brazos protegiéndome) para después decirme en que me había equivocado y como debería de haberlo hecho. Espero que estés orgulloso de mi.

Simplemente gracias por todo lo que hiciste por mí.

Resumen

Este proyecto plantea el desarrollo de un sitio web como herramienta de apoyo al aprendizaje del funcionamiento de los algoritmos.

El aprendizaje del funcionamiento de los algoritmos resulta en ocasiones difícil para los estudiantes. Es fundamental para un Ingeniero en Informática entender en profundidad los mecanismos en los que se basan los algoritmos más conocidos, ya que son la base de la resolución de gran parte de los problemas que es necesario resolver en el ámbito profesional y de investigación. Ejemplos de esquemas algorítmicos importantes son voraz, divide y vencerás, programación dinámica, vuelta atrás y ramificación y poda.

El presente proyecto expone el diseño y desarrollo del framework que se denominó VGA. Este permite una rápida implementación de algoritmos debido a su enfoque orientado a objetos. Asimismo, no se trata de un proyecto cerrado, sino que se diseñó para que su funcionalidad sea aumentada en función de las necesidades.

Actualmente está orientado a la asignatura *"Programación y Estructuras de Datos Avanzadas"*, asignatura del primer cuatrimestre del segundo curso tanto del Grado en Ingeniería Informática como en el Grado en Ingeniería en Tecnologías de la Información de la UNED. No obstante, puede ser objeto de utilización de otras asignaturas por su posible reutilización de objetos para otros menesteres. Ejemplos de estosilustrac pueden ser "Autómatas, Gramáticas y Lenguajes", "Aprendizaje Automático" o ambas asignaturas de inteligencia artificial.

Palabras clave

VGA

Algoritmos

Dijkstra

Dinámica

Estructuras

Grafos

Visualizador

Gráfico

Web

CSS

HTML

JavaScript

BootStrap

GitLab

Framework

English title and abstract

GENERIC ENVIRONMENT FOR THE VISUALIZATION OF THE OPERATION OF ALGORITHMS

This project proposes the development of a website as a support tool for learning how algorithms works.

Learning how algorithms work is sometimes difficult for students. It is fundamental for a Computer Engineer understand in depth the mechanisms on which the most known algorithms are based, since they are the basis for solving many of the problems that need to be solved in the professional and research field. Examples of important algorithmic schemes are greedy, divide and conquer, backtracking and branch and bound.

This project sets out the design and development of the framework that was called VGA. This allows for rapid implementation of algorithms due to its object-oriented approach. It is also not a closed project but was designed to increase its functionality according to needs.

Currently it is oriented to the *subject "Programming and Advanced Data Structures",* course of the first semester of the second year of both the Degree in Computer Engineering and the Degree in Engineering in Information Technologies of UNED. However, it can be used for other subjects because of its possible reuse the objects for other needs. Examples of these can be "Automata, Grammars and Languages", "Automatic Learning" or both artificial intelligence’s courses.

Keywords

VGA

Algorithms

Dijkstra

Dynamic

Structures

Grafts

Viewer

Graphic

Web

CSS

HTML

JavaScript

BootStrap

GitLab

Framework

Índice

Índice de figuras

Índice de tablas

1 Introducción

El presente documento constituye la memoria del Proyecto de Fin de Grado con título *"Entorno genérico para la visualización del funcionamiento de los algoritmos"*.

El objetivo fundamental de este proyecto es ayudar a los alumnos de la asignatura *"Programación y estructuras de datos avanzadas"*, impartida en segundo curso de los Grados en Ingeniería Informática y en Ingeniería en Tecnologías de la Información de la UNED, para poder visualizar el funcionamiento de los algoritmos y sus estructuras internas.

El proyecto consiste en la elaboración de un marco de genérico para la visualización de la ejecución de los algoritmos. Este marco de desarrollo se implementa en un sitio web que se ejecuta en el lado del cliente, permitiendo así que el coste de mantenimiento sea mínimo. Asimismo, se trata de un sistema multiplataforma, por lo que su uso puede ser ampliamente extendido en las labores de la enseñanza de algoritmos, más allá incluso de la asignatura en la que nos hemos centrado.

1.1 Motivación y definición del problema

El aprendizaje del funcionamiento de los algoritmos resulta en ocasiones difícil para los estudiantes. Es fundamental para un Ingeniero en Informática entender en profundidad los mecanismos en los que se basan los algoritmos más conocidos, ya que son la base de la resolución de gran parte de los problemas que es necesario resolver en el ámbito profesional y de investigación. Ejemplos de esquemas algorítmicos importantes son voraz, divide y vencerás, programación dinámica, vuelta atrás y ramificación y poda.

1.2 Alcance y objetivos

El objetivo de este proyecto es desarrollar una herramienta que facilite la comprensión del funcionamiento de los algoritmos presentando de forma visual cómo se modifican las estructuras de datos utilizadas por cada algoritmo. De esta forma, esta herramienta se podrá utilizar como complemento al estudio

teórico del algoritmo de forma que sea fácil de entender los pasos que va dando el algoritmo correspondiente.

Como el sistema mostrará el funcionamiento de los algoritmos presentando los cambios producidos en las estructuras de datos utilizadas, la primera parte del proyecto se centrará en diseñar la visualización de diferentes estructuras (como matrices uni y bidimensionales, árboles, grafos, listas, pilas, colas…), diseñando un API para poder actualizar la vista de las estructuras a medida que se producen los cambios. Además, estas APIs deberán permitir introducir valores iniciales en las estructuras para ejecutar los algoritmos sobre ellos.

La segunda parte del proyecto consistirá en el diseño de un marco genérico en el que se puedan ejecutar algoritmos enriquecidos con llamadas a las APIs de visualización desarrolladas en la primera parte. Se añadirá una estructura de paginación mediante la cual el usuario podrá navegar hacia adelante o atrás a través de los distintos pasos dados por el algoritmo.

Una vez esté disponible el marco genérico de visualización, se alimentará con diferentes algoritmos que ilustrarán el funcionamiento de la aplicación.

Todo esto se implementará en un entorno web, mediante JavaScript, que se ejecutará en el cliente, con el objeto de que esté pueda ser independiente de sistema operativo y dispositivo.

1.3 Metodología de trabajo

Para el desarrollo de este proyecto, se ha adoptado un marco de trabajo denominado BDD (Behaviour Driven Development). BDD es una evolución de TDD (Test Driven Development), que a su vez es un marco de trabajo Agile.

Agile se define como la habilidad de crear y responder a los cambios en el desarrollo[1]. A partir de este principio se desarrollan una serie de prácticas que ayudan a realizar desarrollos que permiten responder de forma ágil a entornos cambiantes. Agile como tal no es un marco de trabajo que pueda utilizarse, si no que es un concepto sobre el que se pueden derivar marcos de trabajo

concretos, como Scrum, Extreme Programming, Feature Driven Development (FDD) o TDD.

BDD [2] surge como un marco que directamente pone el foco en el comportamiento que debe tener la aplicación en función del usuario que la utiliza. Por tanto, se ha adoptado esta metodología básicamente por dos razones:

- Inicialmente había una serie de ideas básicas acerca de lo que se quería obtener, pero faltaban muchos aspectos clave por definir. Por lo tanto, el desarrollo del proyecto debía soportar los cambios. Esto es una de las ventajas fundamentales de las metodologías Agile, sea cual sea el marco de trabajo final utilizado.
- En concreto se ha escogido BDD porque se centra en lo que quieren cada uno de los determinados roles de usuario existentes en la aplicación. Dado que el objetivo principal de este proyecto es ayudar a los alumnos y docentes en su estudio y comprensión, esta metodología encaja a la perfección haciendo de las necesidades del alumno y el docente el motor principal del diseño.

En BDD los requerimientos se definen en forma de historias de usuario, las cuales se componen de un título, descripción y criterios de aceptación [3].

La descripción debe expresarse en los siguientes términos: Como <rol usuario>, quiero <característica a implementar>, para que <justificación de negocio>.

Los criterios de aceptación son las condiciones que se deben cumplir para que el cliente final acepte el trabajo realizado para la historia como hecho. Estos criterios deben definir el qué, pero no el cómo. Aunque no hay ninguna norma específica de cómo construirlos, hay varios formatos aceptados por la comunidad [4]. En concreto, se ha utilizado el orientado a reglas, donde los criterios pueden expresarse en formato lista. Tiene la particularidad de que por cada punto que se describa, debe existir una prueba que lo valide.

1.4 Estructura de la memoria

Del apartado 1.3 se concluye que la documentación debería realizarse por cada historia de usuario, y con su descripción, detalles de su implementación y sus pruebas. Si bien esto sería correcto desde el punto de vista teórico de las metodologías Agile, para el caso concreto de este proyecto se ha considerado práctico, ya que la implementación ha necesitado de constante refactorización debido a la interconexión de muchas de las historias. Es por esto por lo que la memoria se presenta en un formato un poco más clásico, pero siguiendo las buenas prácticas BDD.

En el capítulo 2 hago un resumen de las herramientas que se pueden encontrar en la actualidad y como se acopla a lo necesitado.

En el capítulo 3 explico cuáles son los objetivos de este proyecto. Cuáles son las estructuras de datos y de control básicas que se deben de desarrollar.

En el capítulo 4 especifico las historias de usuario, donde se presentan de forma formal todas las historias que han ido surgiendo durante el desarrollo de la herramienta.

En el capítulo 5 explico que herramientas se han usado para el desarrollo, así como las distintas partes involucradas en la aplicación final.

El capítulo 6 especifica las pruebas realizadas por cada historia de usuario final y el resultado de su ejecución. También se añaden un conjunto de pruebas

Finalmente, en el capítulo 7, expongo mis conclusiones acerca del proyecto y puntos de expansión y mejora.

En los anexos a esta memoria, he añadido tutoriales para conectarse al proyecto, así como un conjunto de ejemplos que amplían la funcionalidad de VGA.

2 Estudio de herramientas existentes

En este capítulo se revisan las características de otras plataformas o herramientas actuales que muestran el funcionamiento de los algoritmos, preferentemente también con fines didácticos, a fin de analizar cuánto se acercan a nuestros objetivos y qué características pueden ser incorporadas al proyecto.

2.1 Análisis

Existen diversas herramientas y plataformas online para el estudio de los algoritmos. Estas incluyen uno o más algoritmos dentro de su catálogo con fines didácticos utilizando HTML y JavaScript.

A continuación, se presenta el análisis de varias de estas plataformas.

2.1.1 VisuAlgo

VisuAlgo[5] se define como un visualizador de estructuras de datos y algoritmos con animación.

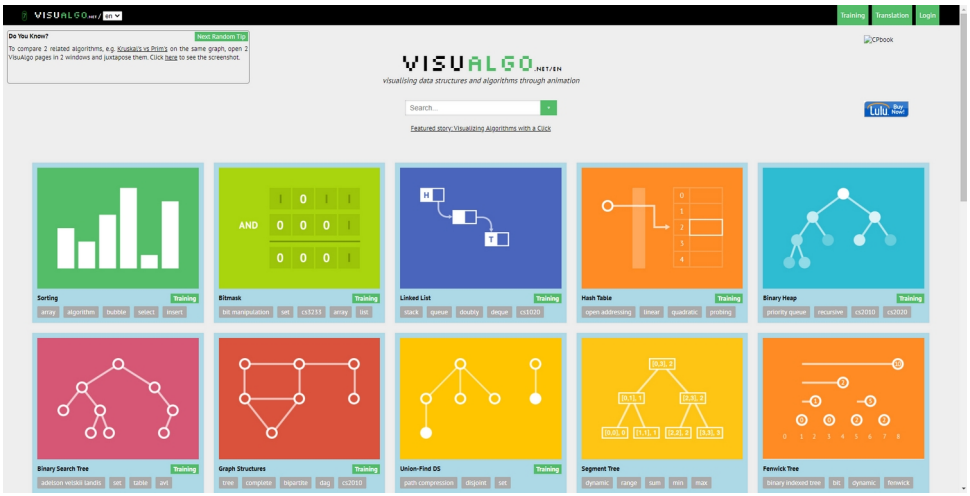


Figura 1: Imagen principal de VisuAlgo

Este proyecto fue iniciado por el Doctor Steven Halim como una herramienta para ayudar a sus estudiantes a entender las estructuras de datos y algoritmos, permitiéndoles aprender los conceptos básicos de estos.

Dentro de esta herramienta se pueden encontrar, en la fecha de la confección de esta memoria, 24 algoritmos de distinta índole. No dispone de ningún tipo de organización de algoritmos más allá de una disposición en cuadricula, en la que cada uno de los diferentes algoritmos posee una colección de tags asociados al mismo.

Si entramos en el algoritmo de ordenación se nos permite elegir el algoritmo a utilizar, así como los datos de entrada. Incluye el pseudocódigo del algoritmo que se está empleando y anima la ejecución de este. Esta ejecución se puede detener, continuar, retroceder y avanzar al gusto del usuario.

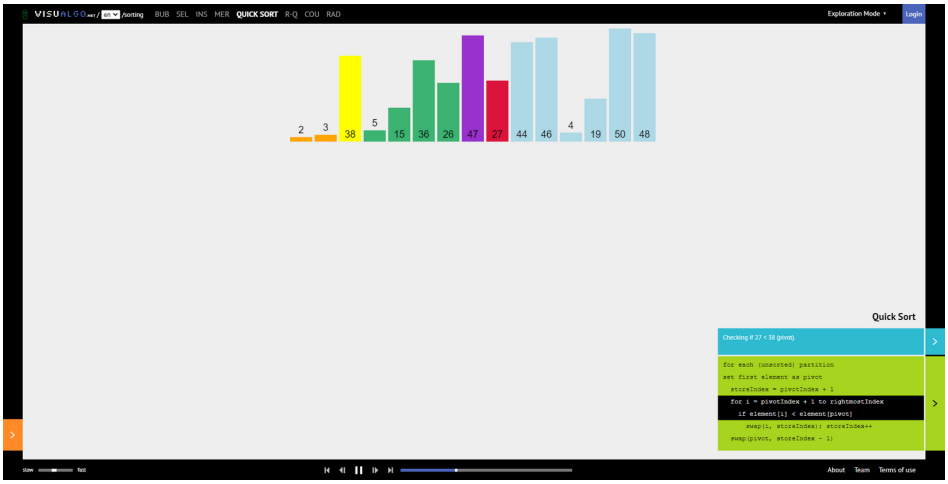


Figura 2: Ejecución del algoritmo Mergesort

Por la contra, esta herramienta carece de una base teórica donde se explique el funcionamiento del algoritmo, el detalle de sus estructuras internas de datos y el coste de este.

Asimismo, se trata de un sistema que es hecho ad-hoc para cada uno de los algoritmos en los que no se tiene un framework que ayude al desarrollador a centrarse en la explicación del algoritmo.

2.1.2 Algorithm Visualizer

Algorithm Visualizer[6] es otra herramienta de visualización de estructuras y algoritmos de código abierto. Este se encuentra alojado en GitHub[7] y es mantenido por la comunidad.

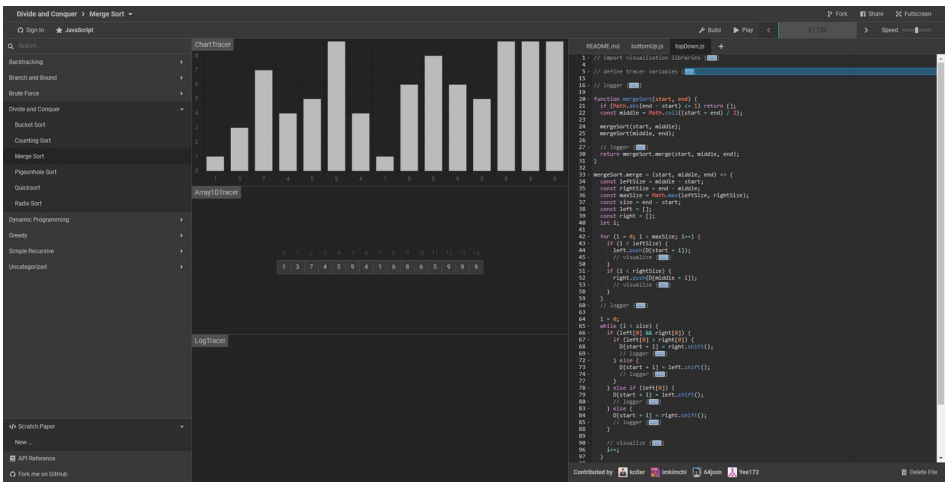


Figura 3: Algorithm Visualizer en el algoritmo de Mergesort

Posee más de 50 algoritmos que son categorizados en función del tipo de estrategia utilizada en en la codificación del algoritmo. Posee la gran peculiaridad de que permite modificar el código del algoritmo, recompilarlo y ejecutar la modificación de este. Al igual que VisualAlgo, se puede controlar los pasos de ejecución del algoritmo y se animan las estructuras de datos y la ejecución del algoritmo.

Cada algoritmo dispone de un archivo README donde se explica brevemente el mismo. Esta documentación varia en contenido y completitud para cada uno de ellos.

Como puntos fuertes, podemos decir que dispone de un framework para la visualización de los algoritmos y su disposición en código abierto facilita su mantenimiento y actualización.

Por la contra, el código que se muestra para la explicación es el usado para la ejecución, por lo que es dependiente del sistema.

Asimismo, esta herramienta está desarrollada en React[8] y Node.js[9], ambos basados en aplicaciones del lado del servidor, con lo que su despliegue se vuelve más laborioso y costoso.

2.1.3 Graph Online

Graph Online [10] es una web donde inicialmente se puede resolver el algoritmo de Dijkstra. Consta de un editor de grafos en los que se puede ir añadiendo nuevos nodos y aristas al grafo y aplicar el algoritmo y algunos más añadidos, como la búsqueda de ciclos hamiltonianos.

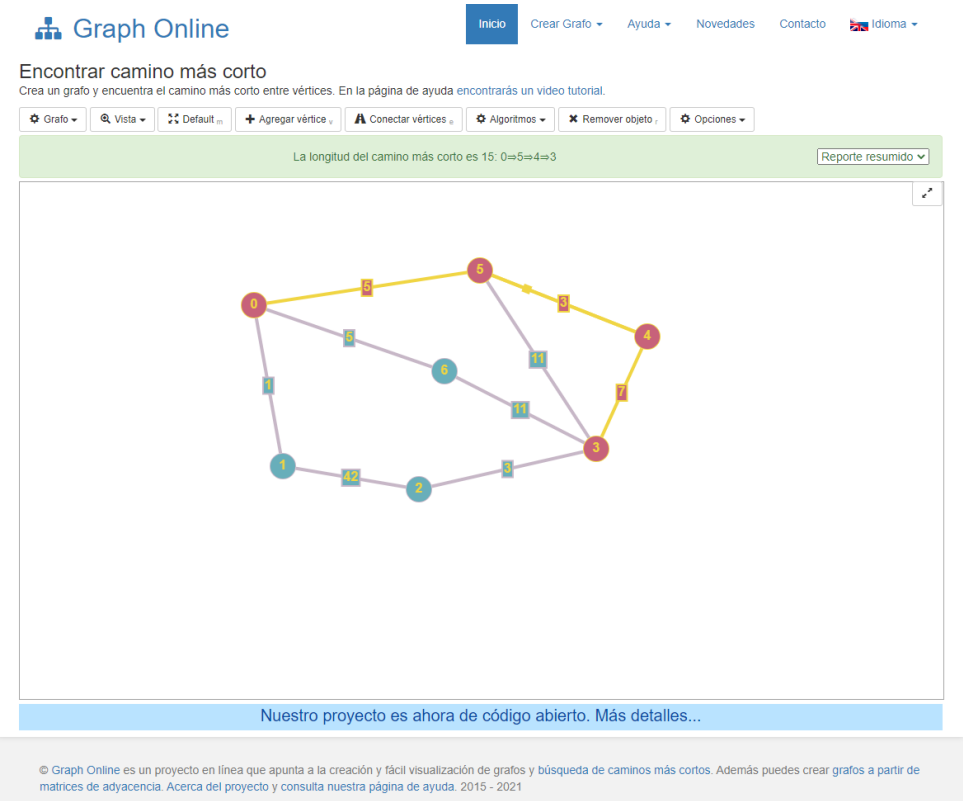


Figura 4: Camino más corto resuelto en Graph Online

Aunque el proyecto es de código abierto, solo posee la capacidad de representar grafos, por lo que su uso es limitado. Asimismo, no se muestra ni una parte teórica del algoritmo ni el pseudocódigo asociado al mismo. Finalmente, no permite ver cómo se ejecuta paso a paso el algoritmo.

2.2 Conclusiones

Aunque las herramientas que se han mostrado en este apartado son una excelente ayuda para la docencia y aprendizaje, es importante ver las características que buscamos en nuestro entorno:

Herramienta	Código abierto	Base teórica	Pseudocódigo	Ej. controlada	Estructuras internas	Ampliable
VisuAlgo	No	No	Sí	Sí	Sí	No, código propietario
Algorith Visualizer	Sí	Depende del algoritmo	Escrito en JS, no	Sí	Sí	Sí
Graph Online	Sí	No	No	No	No	Solo para grafos
VGA	Sí	Sí	Si	Sí	Sí	Sí

Tabla 1: Comparativa de las herramientas analizadas frente a los objetivos de VGA

Con el desarrollo de VGA se obtiene un entorno genérico que permite el desarrollo de algoritmos. Este algoritmo tendrá una base teórica y el pseudocódigo asociado. Este algoritmo se puede simular modificando los datos de entrada y viendo como se ejecuta paso a paso mostrando cómo varían las estructuras de datos.

Aunque este entorno se ha desarrollado inicialmente como asistencia a la asignatura de Programación y Estructuras de Datos, su enfoque genérico permitirá que otras asignaturas puedan desarrollar sus propios algoritmos. Asimismo, podrán añadirse nuevas estructuras de datos al ser estas estructuras orientadas a objetos.

3 Objetivos

El objetivo principal de este proyecto es crear una API que ayude en la creación, manejo y visualización de diferentes tipos de estructuras de datos que se dan en los diferentes algoritmos que el alumno de la UNED se puede encontrar. Ejemplos de estos son matrices unidimensionales y bidimensionales, pilas, colas, grafos y árboles.

Asimismo, será objeto del proyecto crear un entorno web para la presentación tanto de los algoritmos como de sus estructuras internas. La utilización de esta deberá ser lo más amigable y accesible posible. También se procurará que el desarrollo de nuevos algoritmos sea lo más rápido y ágil posible.

3.1 Estructuras de datos

En los siguientes apartados indicaremos los objetivos en cuanto a las diferentes estructuras de datos que que se deberán desarrollar para el presente proyecto.

3.1.1 Matrices unidimensionales

Un matriz unidimensional es un tipo de datos estructurado que está formado por una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos. Todos los datos que almacenan deben ser del mismo tipo y son referenciados mediante un índice de posición, que normalmente empieza en cero.

Índice	0	1	2	3	4
	10	20	30	40	50

Tabla 2: Representación de una matriz unidimensional

Este tipo de estructura también suele recibir el nombre de array o vector. Este tipo de dato se debe representar de dos posibles formas:

1. **Como una tabla.** La representación de una matriz unidimensional se realizaría con la ayuda de algún control especializado en tablas, permitiendo así añadir una cabecera a la misma de manera opcional.

Título columnas	1	2	3	4	5
Título	10	20	30	40	50

Tabla 3: Representación de una matriz unidimensional como tabla

2. **Como una lista.** La representación de la matriz unidimensional puede representarse como una lista continua de los objetos que contiene separados por un objeto delimitador.

10,20,30,40,50

Ambas representaciones ayudarán considerablemente en la representación de este tipo de dato.

Las estructuras deberán permitir la consulta y modificación de los datos. Asimismo, será requisito deseable que permita añadir y eliminar elementos del mismo.

3.1.2 Matrices bidimensionales

Podemos considerar las matrices bidimensionales como una especialización de las unidimensionales. A diferencia de las primeras, las matrices dimensionales poseen dos índices, uno por cada dimensión.

Índice	0	1	2	3	4
0	00	01	02	03	04
1	10	11	12	13	14
2	20	21	22	23	24
3	30	31	32	33	34
4	40	41	42	43	44

Tabla 4: Representación de una matriz bidimensional

Este tipo de estructura queda perfectamente representada mediante una tabla, a la que se le puede añadir cabeceras a las filas y columnas opcionalmente.

Las estructuras deberán permitir la consulta y modificación de los datos. Asimismo, será requisito deseable que permita añadir y eliminar elementos del mismo. La visualización deberá contener mecanismos que permitan la selección gráfica de uno o varios elementos.

3.1.3 Listas

Una lista es una colección de elementos en la que los elementos tienen una posición [11]. Puede pensarse que esta estructura puede considerarse como una especialización de la matriz unidimensional, ya que la única diferencia entre ambas será la capacidad de añadir y eliminar elementos que poseen las listas.

No obstante, se creará una estructura para el manejo de los datos que contiene y la visualización de la lista en formato tabla para que sea más fácil la lectura de esta.

Índice	Título
0	1
1	2
2	3
3	4
4	5

Tabla 5: Representación de una lista

Se podrá consultar y modificar cada uno de los elementos que la componen, al a par de que se añadirá capacidad para añadir y eliminar elementos dinámicamente para representar fielmente la naturaleza de esta estructura.

3.1.4 Pilas y colas

Una pila es una estructura de datos en la que el acceso está restringido al elemento más reciente insertado [11]. Este tipo de estructura también es denominada FIFO (First In, First Out), en el cual el último objeto insertado es el primero en salir. Su analogía es una columna de cajas, la cual siempre cogeremos la última que hemos puesto.

Internamente su estructura es idéntica a una lista, salvo por las peculiaridades que se han indicado.

Por otro lado, esta *la cola, la cual restringe el acceso limitándolo al elemento menos reciente insertado* [11]. Este tipo de estructura también se denomina como LIFO (Last In, First Out), en el cual el primer objeto insertado es el primero en salir. Su analogía en el mundo real es una lista de personas que están esperando para comprar una entrada de cine, se atenderá al primero que ha llegado.

La representación de estas estructuras se realizaran de la siguiente manera:

	Título
4 (Cima)	5
3	4
2	3
1	2
0	1

Tabla 6: Representación de una pila

	Título
4	5
3	4
2	3
1	2
0 (Frente)	1

Tabla 7: Representación de una cola

3.1.5 Grafos

Un grafo es una colección de nodos o vértices unidos por líneas o aristas... Los grafos permiten modelar problemas en los que existe una relación relevante entre los objetos que intervienen. Los nodos o vértices representarían los objetos y las aristas las relaciones entre ellos.[12]

Se trata de una estructura de la que se basan muchos de los algoritmos que se pueden encontrar en los distintos libros académicos relacionados con la programación y algoritmia.

A continuación se muestra un ejemplo de un grafo:

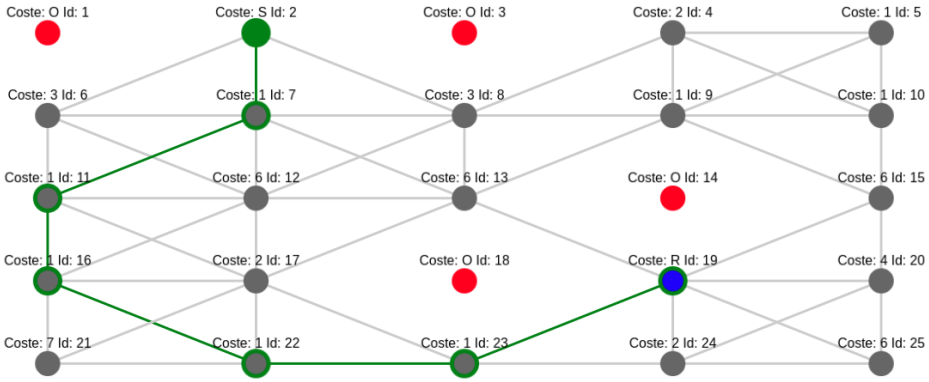


Figura 5: Representación de un grafo

Las estructuras deberán permitir la consulta y modificación de los datos. Asimismo, será requisito deseable que permita añadir y eliminar elementos del mismo. La visualización deberá contener mecanismos que permitan la selección gráfica de uno o varios elementos.

3.1.6 Árboles

Un árbol es un grafo en el que cualquier par de vértices están conectados por exactamente un camino, o alternativamente, es un grafo conexo acíclico. Por tanto, un árbol es una especialización de un grafo. Aunque un árbol puede tener la forma que se considere oportuna, es deseable que se pueda obtener una representación como la que sigue.

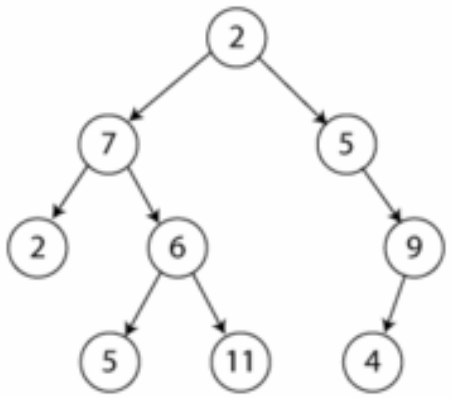


Figura 6: Representación de un árbol

Asimismo, seguirá teniendo todas funcionalidades de consulta, modificación y personalización que dispone la estructura de grafo.

3.1.7 Gráficas

Inicialmente se pretende que las gráficas tengan un enfoque simple para la ayuda de representar algoritmos de ordenación. Aunque se trate de una estructura basada en una matriz uni o bidimensional es conveniente considerarlo como una estructura independiente debido a su complejidad interna.

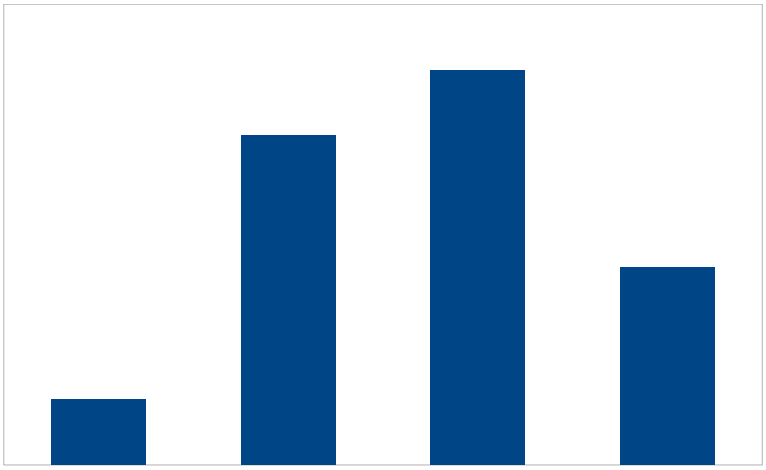


Figura 7: Representación de una gráfica

Esta gráfica debe permitir consultar y modificar datos, así como la adición, eliminación y la selección de los datos que representa.

3.2 Representación del pseudocódigo

Para una correcta explicación del código es importante poder seguir la ejecución del algoritmo. Para ello, deberá realizarse un control que permita imprimir el pseudocódigo del algoritmo y seleccionar las líneas de ejecución actuales. Este pseudocódigo no deberá ser dependiente de ningún lenguaje, por lo que se dispondrá de mecanismos para definir las palabras clave de las que dispone el lenguaje usado por el desarrollador.

```
tipo VectorNat = matriz[0..n] de natural

fun Dijkstra (G = : grafo, R: natural, S: natural): VectorNat, VectorNat
  var
    especial, predecesor: VectorNat
    C: conjunto de nodos
  fvar
  C={1,2,3,...,n} excepto R
  para i = 1 hasta n y i ≠ R hacer
    especial[i] = Distancia(R ,i)
    predecesor[i] = R
  fpara
  mientras C contenga al nodo S hacer
    v = nodo en C que minimiza especial[v]
```

Figura 8: Ejemplo de representación de pseudocódigo

3.3 Control de la ejecución del algoritmo

Para la explicación del algoritmo como de su entendimiento, es importante que la ejecución del algoritmo se pueda pausar y reanudar.

Para ello se elaborará un sistema de historia donde se guardara la ejecución de los pasos que ejecuta el algoritmo, pudiendo reproducir, pausar, avanzar y retroceder un paso o hasta el inicio/final del mismo.

Estos controles deberán ser visibles en todos los puntos de la página, debido a que un algoritmo puede ser lo suficientemente extenso y tener abundantes estructuras de datos.

3.4 Entorno de visualización y ejecución de algoritmos

El sistema deberá ser portable a cualquier sistema operativo y dispositivo.

Para ello, se realizará un sitio web en el que se hará uso exclusivamente de paginas html, archivos de estilo y archivos de JavaScript. En ningún momento se utilizará ninguna tecnología que implique un procesamiento de datos en servidor, teniendo que ejecutarse unicamente en la parte cliente.

Este enfoque permitirá que el alojamiento necesario para VGA sea lo más simple y económico posible. Asimismo, contribuirá a que la distribución de VGA sea más fácil al no tener la necesidad de instalar otro software que no sea un servidor web para la plataforma en la que se desee instalar.

4 Historias de usuario

Como se ha indicado en el apartado 1.4, se han ido recopilando las historias de usuario durante el desarrollo del proyecto. En este capítulo se muestran estas historias recopiladas.

En la figura se muestra un diagrama resumen de todas y cada una de las historias de usuario que se han definido para el proyecto. Puede observarse la existencia de dos actores. El usuario final hace referencia al consumidor de VGA, mientras que el usuario desarrollador lo hace hacia aquel actor cuya función consiste en mantener el proyecto con nuevos controles y algoritmos.

En los siguientes subcapítulos se ordenan las historias para cada uno de los actores indicados.

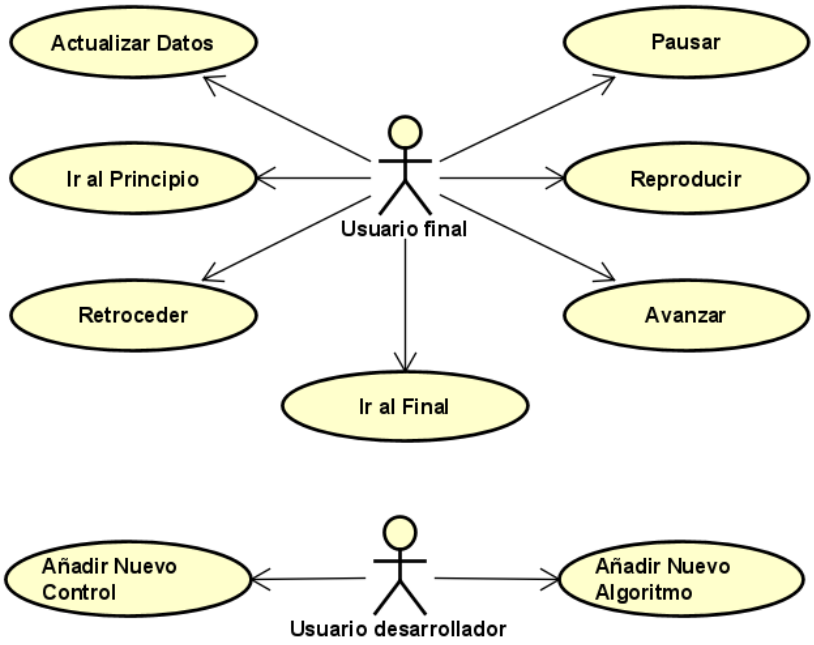


Figura 9: Diagrama general de Historias de Usuario

4.1 Historias de usuario final

A continuación se detallan las historias asociadas al usuario final de la aplicación.

4.1.1 Historia HU001 – Actualizar datos

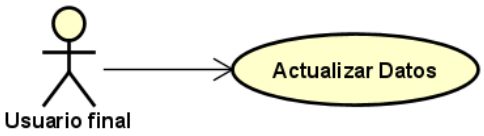


Figura 10: Actualizar datos

4.1.1.1 Descripción

Como usuario final, quiero poder modificar los datos de entrada del algoritmo, de tal manera que se actualice la ejecución del algoritmo con los datos que he introducido.

4.1.1.2 Criterios de aceptación

- El usuario puede modificar los datos de entrada según los parámetros de entrada.
- Los parámetros de entrada se ajustarán en función del tipo de dato:
    - Si es un elemento simple, se permitirá la modificación en una caja de texto.
    - Si es un vector o matriz, se permitirá su modificación en una tabla, pudiendo, o no, añadir y eliminar más filas en función del parámetro.
- El algoritmo se ejecuta una vez que el usuario pulsa el botón de “Actualizar datos”.
- El algoritmo vuelve al primer paso una vez se ejecuta.
- El algoritmo sitúa el cursor de código en la primera línea.

4.1.2 Historia HU002 – Ir al principio

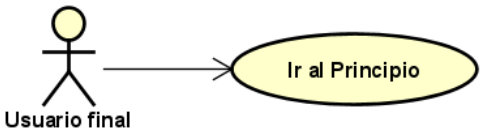


Figura 11: Ir al principio

4.1.2.1 Descripción

Como usuario final, quiero poder volver al primer paso del algoritmo, pudiendo así volver a ejecutarlo desde el principio.

4.1.2.2 Criterios de aceptación

- Se vuelve al primer paso del algoritmo.
- Se reinician los controles de datos de la salida.
- El algoritmo sitúa el cursor de código en la primera línea.
- Si el algoritmo se encuentra en el primer paso no se realiza ninguna acción.

4.1.3 Historia HU003 – Retroceder

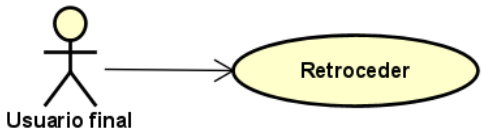


Figura 12: Retroceder

4.1.3.1 Descripción

Como usuario final, quiero poder retroceder un paso en la ejecución del algoritmo, devolviendo los datos a su estado anterior.

4.1.3.2 Criterios de aceptación

- Se retrocede un paso en la ejecución.
- Se deshacen los cambios realizados por el paso que se retrocede.
- El algoritmo sitúa el cursor de código en la línea anterior.
- Si se encuentra en el primer paso del algoritmo no se realiza ninguna acción.

4.1.4 Historia HU004 – Reproducir

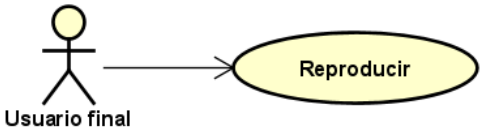


Figura 13: Reproducir

4.1.4.1 Descripción

Como usuario final, quiero poder ejecutar el algoritmo de manera continua, de tal manera que automáticamente se continúe al siguiente paso cada periodo de tiempo.

4.1.4.2 Criterios de aceptación

- Se debe avanzar al siguiente paso tras el transcurso de un segundo.
- Se debe de avanzar desde el paso actual en el que se encuentra el algoritmo.
- Se debe detener la reproducción cuando se llegue al último paso.
- Se deben actualizar los datos en cada paso ejecutado.
- Se debe de actualizar el cursor de código en la línea ejecutada.
- Si se está en el último paso no se realizará ninguna acción.
- Sólo visible si el estado actual es no reproduciendo.

4.1.5 Historia HU005 – Pausar

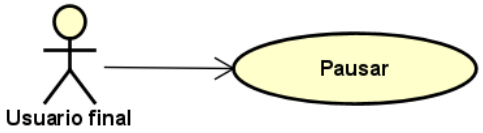


Figura 14: Pausar

4.1.5.1 Descripción

Como usuario final, quiero poder detener la reproducción automática, de tal manera que la ejecución se detenga en el paso actual.

4.1.5.2 Criterios de aceptación

- Se detiene la reproducción automática.
- Se detiene en el paso actual.

- Sólo visible si el algoritmo se encuentra reproduciéndose.

4.1.6 Historia HU006 – Avanzar

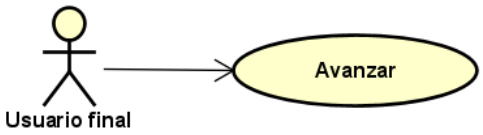


Figura 15: Avanzar

4.1.6.1 Descripción

Como usuario final, quiero poder avanzar el algoritmo un paso, pudiendo así ver la modificación de las estructuras a causa de la ejecución del paso.

4.1.6.2 Criterios de aceptación

- Se avanza un paso en la ejecución del algoritmo.
- Se actualizan los datos de salida con los datos que el algoritmo ha modificado.
- Se actualiza el cursor de código a la línea correspondiente.
- Si se encuentra en el último paso no se realiza ninguna acción.

4.1.7 Historia HU007 – Ir al final

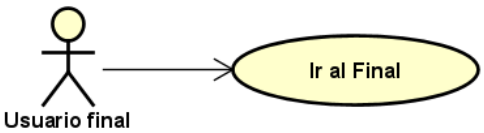


Figura 16: Ir al final

4.1.7.1 Descripción

Como usuario final, quiero poder ir al último paso del algoritmo, pudiendo así ver el resultado de ejecutar el algoritmo.

4.1.7.2 Criterios de aceptación

- Se avanza al último paso del algoritmo.
- Se modifican los controles de datos de la salida con los datos resultantes de la ejecución de código.
- El algoritmo sitúa el cursor de código en la última línea.

- Si el algoritmo se encuentra en el primer paso no se realiza ninguna acción.

4.2 Historias de usuario desarrollador

A continuación se detallan las historias asociadas al usuario desarrollador de la aplicación.

4.2.1 Historia HU008 – Añadir nuevo control

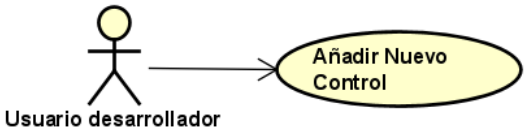


Figura 17: Añadir nuevo control

4.2.1.1 Descripción

Como usuario desarrollador, quiero un marco de trabajo con el que se puedan añadir nuevos controles a la herramienta.

4.2.1.2 Criterios de aceptación

- El framework tiene una orientación a objetos para que exista una abstracción en los objetos.
- Se preparan objetos primitivos sobre los que heredar los nuevos controles.

4.2.2 Historia HU009 – Añadir nuevo algoritmo

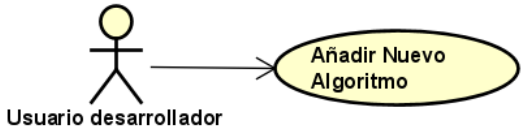


Figura 18: Añadir nuevo algoritmo

4.2.2.1 Descripción

Como usuario desarrollador, quiero una plantilla con la que pueda realizar un nuevo algoritmo para la herramienta.

4.2.2.2 Criterios de aceptación

- Existe una plantilla html para la capa de presentación y control del algoritmo.
- Existe una plantilla JavaScript con los elementos básicos para la ejecución del algoritmo.

4.3 Requisitos no funcionales

En cuanto a su ejecución, se deberá de realizar en el lado del cliente (frontend). Esto implica que todos los cálculos y operaciones que dispondrá la aplicación deberán ejecutarse en la computadora del consumidor de la aplicación. Este enfoque permitirá que el despliegue de VGA sea más sencillo.

Como requisito económico se marcará como objetivo que sea lo menor posible. Esto va de la mano del requisito de la ejecución en frontend dado que, al no necesitarse requisitos de backend (lado del servidor) importantes, podrá alojarse en un hosting básico.

5 Diseño y construcción

En este capítulo vamos a analizar diferentes tecnologías que nos permitirían implementar nuestra aplicación, para poder seleccionar las más adecuadas a nuestros objetivos.

5.1 Tecnología utilizada

En el capitulo 2 hemos visto las diferentes herramientas existentes analizadas, de las cuales la inmensa mayoría se ejecutaba del lado del servidor.

Podríamos pensar en usarlo para VGA, teniendo aquí dos posibilidades:

- **Spring:** Se trata de un framework que utiliza la tecnología J2EE y volviéndola más simple de desarrollar al basarse en buenos principios de desarrollo. Este se utiliza ampliamente y permitiría una programación más amigable al programarse en Java.
- **Node.js + React:** Node.js es un entorno en tiempo de ejecución en el lado del servidor que se programa en JavaScript y React es una biblioteca que permite diseñar aplicaciones en una sola página (basándose en Node.js). En combinación, se lograría una aplicación con una alta capacidad de procesamiento de algoritmos y su capacidad de simular el dinamismo de la aplicación.

No obstante, ambas soluciones se basan en un enfoque de ejecución en el lado del servidor. Este enfoque no cumpliría el requisito funcional de la ejecución del lado del consumidor ni el objetivo del coste de mantenimiento del sistema.

Por consiguiente, se estima que la mejor tecnología a utilizar por el sistema es un sitio web de páginas estáticas con una API propia realizada en JavaScript. Basándose en librerías como jQuery, se realizaran animaciones y modificaciones de datos en el lado del cliente librando de este trabajo al servidor, limitándose unicamente a servir estos datos estáticos.

5.2 Librerías usadas

Para el desarrollo de la API se han utilizado diferentes librerias, las cuales son de amplio uso y mantenimiento por parte de la comunidad.

5.2.1 Bootstrap

Bootstrap[13] es un framework desarrollado por Twitter. Es una biblioteca multiplataforma de código abierto para el diseño de sitios y aplicaciones web. Contiene plantillas de muchos componentes basados en HTML y CSS, así como extensiones JavaScript adicionales. Este framework sólo se ocupa de la parte front-end.

Figura 19: Logotipo de Bootstrap

Podemos considerar Bootstrap como un framework de obligado uso al ayudar a la elaboración de UI con un carácter moderno, a la par de preparado para multiplataforma. En VGA se utiliza para la creación de elementos base como pueden son la barra de menú, los cuadros de texto con sus etiquetas, etc.

5.2.2 jQuery

jQuery[14] es una biblioteca multiplataforma, creada inicialmente por John Resig, que simplifica la interacción con los distintos elementos HTML de una página web. Se trata de una biblioteca de código abierto y está escrita es JavaScript.

Figura 20: Logotipo de jQuery

Al igual que Bootstrap, se trata de un pilar básico para el desarrollo de sitios web que sean reactivos y, dado la dinamicidad de VGA, se utiliza para

manipulación de elementos HTML, así como la consulta de los datos que estos contienen.

5.2.3 Cytoscape.js

Cytoscape.js[15] es una librería de código abierto para la teoría de grafos escrita en JavaScript. Esta librería se puede usar para el análisis y visualización de grafos.

Figura 21: Logotipo de Cytoscape

En VGA se utiliza para la representación de grafos debido a su facilidad de uso y adecuación a las necesidades de los algoritmos que utilizan dicha estructura.

5.2.4 D3.js

D3.js[16] es una biblioteca de JavaScript para la visualización de datos. Hace uso de los distintos estándares como son SVG, HTML5 y CSS3. D3.js permite tener el control completo sobre el resultado visual final.

Figura 22: Logotipo de D3.js

En VGA se utiliza para la representación de un gráfico de barras simple, pero su potencia permitirá su uso en distintos elementos visuales futuros, como pueden ser grafos y objetos complejos con un enfoque dinámico.

En un primer momento, fue candidato para el su uso como librería para la representación de grafos. Sin embargo, se desestimó su uso, a favor de Cytoscape, debido a la complejidad de uso para la manipulación de los grafos.

5.3 Herramientas

Para el desarrollo de la API, así como las páginas que componen el sitio web y la documentación, se ha utilizado el software indicado en esta sección.

5.3.1 Visual Studio Code

Visual Studio Code [17] es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Es un software gratuito y de código abierto con licencia MIT [18].

Este editor tiene la ventaja de que puede usarse para cualquier lenguaje, gracias a su diseño basado en plugins. Sólo es necesario instalar aquellos necesarios para el desarrollo del lenguaje requerido. Para el desarrollo de JavaScript ya está soportado nativamente.

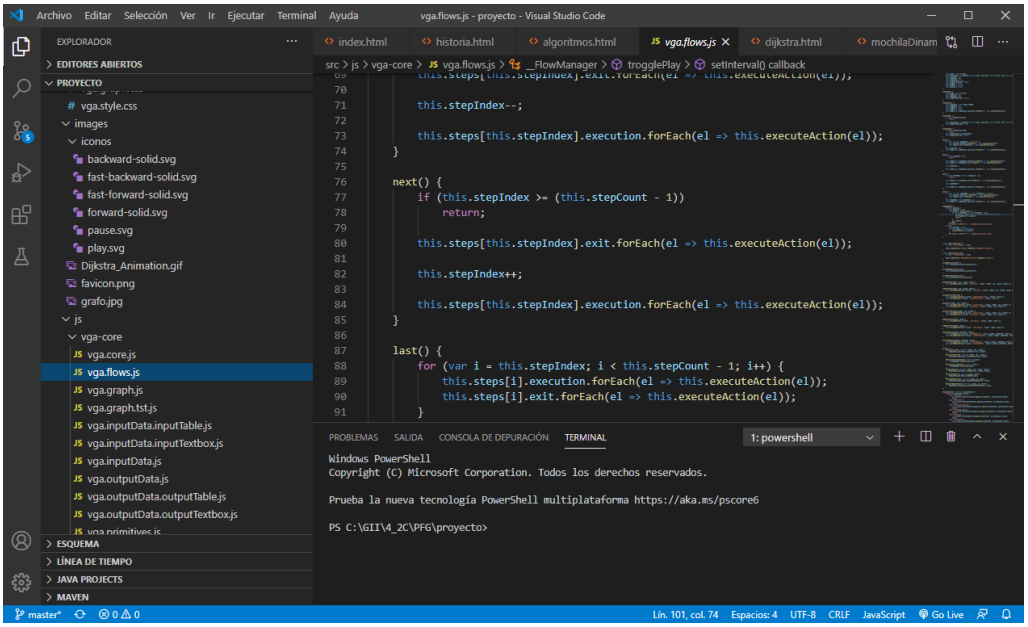


Figura 23: Interfaz de Visual Studio Code

Esta herramienta se ha utilizado para la codificación de la API de VGA y el sitio web.

5.3.2 Brackets

Brackets[19] es un editor de código fuente con un enfoque principal en el desarrollo web. Creado por Adobe Systems, es un software gratuito y de código abierto con licencia MIT [18]

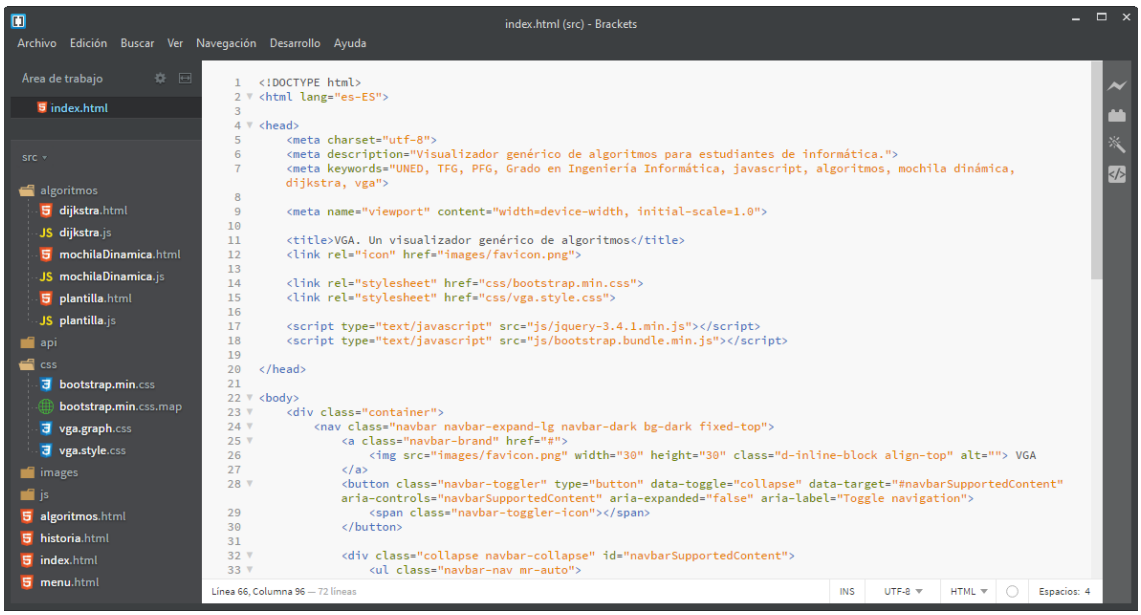


Figura 24: Interfaz de Brackets

Esta herramienta se ha utilizado para el refinamiento de la visualización del sitio web, gracias a su vista previa dinámica.

5.3.3 Atash UML

Atash UML[20] es una herramienta de modelado UML creada por ChangeVision. Es software propietario, pero dispone de una licencia gratuita para estudiantes.

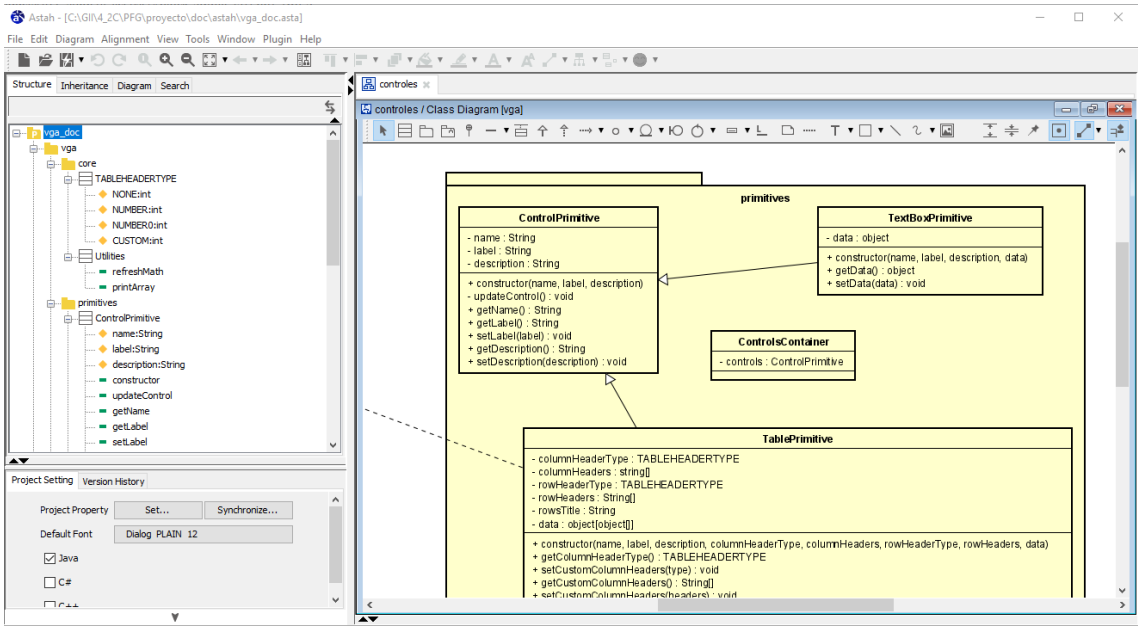


Figura 25: Interfaz de Astah UML

Con esta herramienta se pueden realizar diagramas de casos de uso, de secuencia, de clases... utilizados para el análisis y diseño tanto de la API como del sitio web y su interacción con el usuario.

5.3.4 GitLab

GitLab[21] es un servicio web de control de versiones y desarrollo de software basado en Git. Asimismo, ayuda a la gestión de los proyectos con seguimiento de errores y tareas, así como integración continua.

El desarrollo de VGA se encuentra alojado en este servicio ya que presenta una opción gratuita sin límite de usuarios.

Figura 26: Logotipo de GitLab

5.4 Planificación

Esta planificación se elabora tras comenzar a abordar el Análisis del sistema del ciclo de vida inicial, es decir, podemos ubicarla en la fase de elaboración del ciclo de vida del proyecto.

Se realiza una planificación, repartiendo las diferentes actividades entre las iteraciones que tienen lugar en cada fase del ciclo de vida.

Cada iteración permite obtener un incremento de software, una ampliación sobre el incremento

anterior, más cercano al producto final. Se realiza una asignación de los objetivos principales que se espera cumplir en cada iteración. Esta distribución de las actividades se recoge en la siguiente tabla.

Fase	Iteración	Objetivos
Inicio	0	• Describir el sistema. • Estudio herramientas existentes. • Toma de los requisitos principales. • Diseñar diagrama de casos de uso. • Escritura de casos de uso. • Elección de software y tecnologías. • Planificación del proyecto. • Estimación del coste. • Redacción del anteproyecto.
Elaboración	1	• Creación del repositorio en GitLab • Creación estructura de directorios del proyecto • Obtención de las librerías necesarias
	2	• Elaboración de los archivos de referencia (plantillas) • Elaboración de la clase controladora de pseudocódigo
	3	• Elaboración de las clases controladoras de controles de entrada y salida

	4	• Elaboración del control de entrada y salida de texto • Elaboración del control de entrada y salida de tabla
	5	• Elaboración del control de grafos
	6	• Elaboración del control de gráfico de barras
	7	• Elaboración de la clase controladora del flujo del algoritmo
	8	• Elaboración algoritmo de mochila dinámica • Elaboración algoritmo de Dijkstra • Elaboración de algoritmo Quicksort
	9	• Pruebas de integración • Refinamientos
Documentación PFG	10	• Elaboración de la memoria • Elaboración de las presentaciones
Presentación PFG	11	• Presentación del proyecto

Tabla 8: Fases e iteraciones del proyecto

Una vez distribuidos los objetivos entre las iteraciones y fases, se elaboran diagramas de Gantt que muestran las fechas en las cuales tienen lugar dichas iteraciones y fases.

La fase inicial se compone de una única iteración. Comienza el 5 de abril de 2021 y termina el 7 de mayo de 2021.

- Iteración 0. Comienza el 5 de abril de 2021 y termina el 7 de mayo de 2021.

La fase de elaboración se compone de nueve iteraciones. Comienza el 10 de mayo de 2021 y termina el 30 de julio de 2021.

- Iteración 1. Comienza el 10 de mayo de 2021 y termina el 11 de mayo de 2021.

- Iteración 2. Comienza el 12 de mayo de 2021 y termina el 14 de mayo de 2021.
- Iteración 3. Comienza el 17 de mayo de 2021 y termina el 28 de mayo de 2021.
- Iteración 4. Comienza el 31 de mayo de 2021 y termina el 4 de junio de 2021.
- Iteración 5. Comienza el 7 de junio de 2021 y termina el 15 de junio de 2021.
- Iteración 6. Comienza el 16 de junio de 2021 y termina el 25 de junio de 2021.
- Iteración 7. Comienza el 28 de junio de 2021 y termina el 7 de julio de 2021.
- Iteración 8. Comienza el 8 de julio de 2021 y termina el 23 de julio de 2021.
- Iteración 9. Comienza el 26 de julio de 2021 y termina el 30 de julio de 2021.

La fase de elaboración de la documentación del PFG y las diapositivas consta de una sola iteración. Comienza el 2 de agosto de 2021 y termina el 20 de agosto de 2021.

- Iteración 10. Comienza el 2 de agosto de 2021 y termina el 20 de agosto de 2021.

La fase de presentación del PFG consta de una sola iteración. Comienza el 23 de agosto de 2021 y termina el 27 de agosto de 2021.

- Iteración 11. Comienza el 23 de agosto de 2021 y termina el 27 de agosto de 2021.

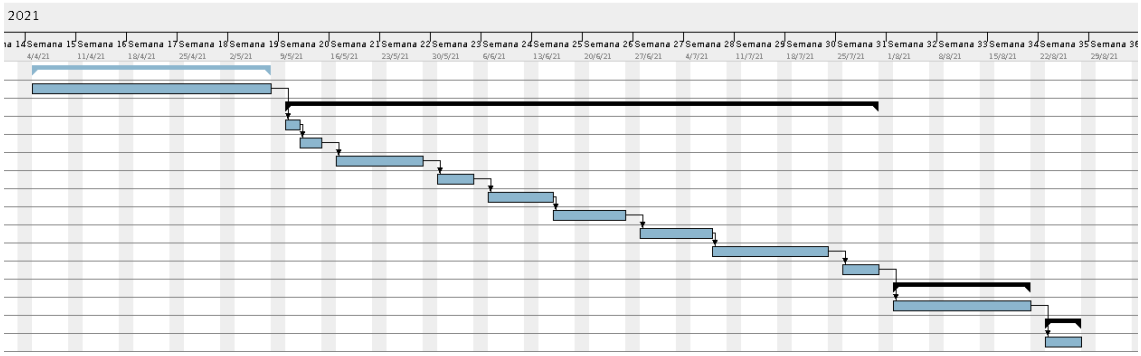


Figura 27: Diagrama de Gantt del proyecto

5.5 Estudio de coste estimado de desarrollo

Antes de comenzar un proyecto es útil realizar una estimación del coste para analizar la viabilidad del mismo. En esta sección, se realizará una estimación del coste empleando la ecuación del software de Putnam [22], como podría hacerse si se pretendiera comercializar el producto.

La ecuación del software es un modelo dinámico multivariable que supone una distribución de esfuerzo específica durante la vida de un proyecto de desarrollo de software.

		Factor ponderado			**Conteo**		
	Conteo	**Simple**	**Promedio**	**Complejo**	**Simple**	**Promedio**	**Complejo**
EEs	7	3	4	**6**	21	28	**42**
SEs	5	4	**5**	7	20	**25**	35
CEs	0	3	**4**	6	0	**0**	0
ALIs	13	7	**10**	15	91	**130**	195
AIEs	0	5	**7**	10	0	**0**	0
Conteo total para casos simple, promedio y complejo					132	183	272
Conteo total considerando diferentes factores de complejidad							**197**

Tabla 9: Estimación de puntos de función

Para realizar la estimación, se seguirá el método del cálculo de los puntos de función. En este método, se construye una tabla con el conteo estimado de las entradas externas (EE), salidas externas (SE), consultas externas (CE), número de archivos lógicos internos (ALI) y número de archivos de interfaz externos (AIE).

Este conteo de los valores del dominio de información se ponderan en base a su complejidad, para obtener un valor del número de puntos de función . Esta determinación de la complejidad puede resultar un tanto subjetiva, pero se pretende realizar de la manera más objetiva posible para afinar la estimación.

Este número de puntos de función calculado se ponderan posteriormente en base a valores de ajuste de valor, que vienen dados por la valoración de diferentes características del proyecto.

Para calcular finalmente los puntos de función se emplea la siguiente ecuación:

$$PF = conteoTotal \times [0{,}65 + 0{,}01 \times \sum_{i=1}^{14} F_i]$$

En dicha ecuación, $\sum_{i=1}^{14} F_i$ se corresponde a los ajustes de valor, que se calculan valorando, en una escala de 0 (no aplicable o no relevante) a 5 (esencial), la respuesta a diferentes cuestiones. Las cuestiones y su valoración son:

1. ¿El sistema requiere respaldo y recuperación confiables? **0 puntos**. No es crítico disponer de un sistema de respaldo para el proyecto, por lo que no se le otorga puntos a esta característica.
2. ¿Se requieren comunicaciones de datos especializadas para transferir información hacia o desde la aplicación? **0 puntos**. Las comunicaciones que se requieren no implican protocolos especiales de comunicación.
3. ¿Existen funciones de procesamiento distribuidas? **1 punto**. Sí, el software debe ser distribuido, y accesible desde Internet, aunque se

delegará en el cliente la mayor parte de la ejecución, por lo que sólo se le otorga un punto.

4. ¿El desempeño es crucial? **3 puntos**. Sí, es importante, aunque no crítico, que tenga un buen rendimiento. Por ello se le otorga tres puntos sobre cinco.

5. ¿El sistema correrá en un entorno operativo existente enormemente utilizado? **5 puntos**. El sistema se pretende ejecutar sobre diferentes dispositivos, plataformas, configuraciones y pantallas (teléfonos, tablets, portátiles, equipos de sobremesa). Por tanto, dada su adaptabilidad se le otorga cinco puntos a esta característica.

6. ¿El sistema requiere entrada de datos en línea? **2 puntos**. Sí, la entrada de datos la realizará el alumno en los campos establecidos para tal uso.

7. ¿La entrada de datos en línea requiere que la transacción de entrada se construya sobre múltiples pantallas u operaciones? **0 puntos**. Más bien al contrario, se pretende que la entrada sea sencilla y se haga a ser posible desde un único formulario.

8. ¿Los ALI se actualizan en línea? 0 puntos. No es relevante para el proyecto, ya que los datos no van a persistirse.

9. ¿Las entradas, salidas, archivos o consultas son complejos? **1 punto**. No, las entradas se intentan que sean lo más simples posibles para facilitar la usabilidad de la herramienta.

10. ¿El procesamiento interno es complejo? **5 puntos**. Sí, el proceso de gestión de las estructuras internas es complejo.

11. ¿El código se diseña para ser reutilizable? **4 puntos**. Sí, se pretende que el software sea legible y se facilite el mantenimiento, por lo que se diseña para que pueda ser reutilizable.

12. ¿La conversión y la instalación se incluyen en el diseño? **0 puntos**. No se considera relevante.

13. ¿El sistema se diseña para instalaciones múltiples en diferentes organizaciones? **0 puntos.** No, se diseña para la UNED.

14. ¿La aplicación se diseña para facilitar el cambio y su uso por parte del usuario? **0**. Sí, se pretende que la aplicación sea fácil de usar, y fácil de mantener.

En total, se disponen de **21 puntos de ajuste de valor**.

Estimado el número de puntos de ajuste de valor, es posible realizar el cálculo de los puntos de función.

$$PF = 197 \times [0{,}65 + 0{,}01 \times \sum_{i=1}^{14} F_i]$$
$$PF = 197 \times [0{,}65 + 0{,}01 \times 21]$$
$$PF = 197 \times [0{,}86] = 169{,}42 \approx 170$$

Una vez estimado el número de puntos de función, es necesario proceder a una estimación del número de LOC (Lines Of Code) que tendrá el proyecto, ya que la ecuación del software de Putnam emplea esta medida.

El número de líneas de código a partir del número de puntos de función depende del lenguaje que se emplee para codificar el software. Se estima que se compondrá de un 90 % de código JavaScript y un 10 % de código HTML y CSS.

En base a los resultados de “Function Point Languages Table” aportados por QSM, en su versión 5.0 [23], se obtienen los siguientes valores, tomando la media de líneas de código por cada uno de estos lenguajes:

1. JavaScript. 47 líneas de código / punto de función.
2. HTML. 34 líneas de código / punto de función.

No existen datos en dicha tabla para el lenguaje CSS, pero dada su similitud con el lenguaje HTML, no se espera que el número de líneas de código por punto de función difiera demasiado, por lo que se aproxima a tomar el mismo número de líneas de código por punto de función que para el lenguaje HTML.

Por tanto, ponderando en base al porcentaje esperado de cada lenguaje, se obtiene la estimación de las líneas de código del proyecto:

$$LOC = puntosDeFunción \times (0{,}9 \times Avg(JavaScript) + 0{,}1 \times Avg(HTML, CSS))$$
$$LOC = 170 \times (0{,}9 \times 47 + 0{,}1 \times 34)$$
$$LOC = 170 \times 45{,}7 = 7769$$

Una vez estimado el número de líneas de código, se procede a calcular el tiempo mínimo de desarrollo y el esfuerzo en persona-mes. Las ecuaciones para realizar estas estimaciones son:

$$t_{min} = 8{,}14 \times \left(\frac{LOC}{P}\right)^{0{,}43}$$
$$E = \left(\frac{LOC \times B^{0{,}333}}{P}\right)^{3} \times \frac{1}{t^4}$$

Para obtener el resultado de ambas ecuaciones es necesario determinar un valor para los parámetros P y B. A continuación se explica el significado de cada uno de estos parámetros y el valor elegido.

1. P. Parámetro de productividad, que refleja: madurez global del proceso y prácticas administrativas, la medida en la que se usan buenas prácticas de ingeniería de software, el nivel de lenguajes de programación utilizado, el estado del entorno de software, las habilidades y experiencia del equipo de software y la complejidad de la aplicación. Los valores típicos pueden ser P = 2000 para desarrollo de un software incrustado en tiempo real, P = 10000 para software de telecomunicaciones y sistemas y P = 28000 para aplicaciones de sistemas empresariales [22]. Se decide asignar un valor de 17.000, teniendo presente que es un software que reúne características que podrían estar presentes en sistemas empresariales y en software de sistemas.

2. B. Factor de habilidades especiales. Al tratarse de un proyecto pequeño, se le asigna el valor de 0,16 [22].

Ahora sí, se procede al cálculo del tiempo mínimo de desarrollo:

$$t_{min} = 8{,}14 \times \left(\frac{LOC}{P}\right)^{0{,}43}$$

$$t_{min} = 8{,}14 \times \left(\frac{7769}{17000}\right)^{0{,}43} = 5{,}8\ meses$$

Y seguidamente, al cálculo del esfuerzo en persona-año, teniendo en cuenta que 5,8 meses ≈ 0,48 años.

$$E = \left(\frac{LOC \times B^{0{,}333}}{P}\right)^{3} \times \frac{1}{t^4}$$

$$E = \left(\frac{7769 \times 0{,}16^{0{,}333}}{17000}\right)^{3} \times \frac{1}{0{,}48^4}$$

$$E = 0{,}24^3 \times \frac{1}{0{,}48^4} = 0{,}288\ persona - año$$

Multiplicando por 12, que son los meses de un año, se puede obtener el cálculo en persona – mes, y redondear al valor entero más cercano.

$$E = 0{,}288 \times 12 = 3{,}46\ persona - mes$$

El esfuerzo obtenido tras redondear el valor, corresponde a 3,5 persona - mes. Suponiendo una tarifa de mano de obra pondera de 5.000 euros / mes, el coste de personal para el desarrollo de la aplicación resulta de multiplicar el esfuerzo en persona-mes por la tarifa de mano de obra:

$$C = 3{,}5\ persona - mes \times 5000\ euros/mes = 17500\ euros$$

Para obtener la estimación del coste total del proyecto, debe añadirse también el coste de la adquisición del software a emplear. Dado que es gratuito, el coste del proyecto bajo las consideraciones anteriores se estima en 17.500 euros.

5.6 Estudio de coste de mantenimiento

Como se ha comentado anteriormente, el sistema sería posible hospedarlo en un servidor de alojamiento. Este alojamiento puede ser básico, dado que VGA no requiere ningún requerimiento especial más hallá que un sitio web de elementos estáticos.

En este estudio de coste se ha realizado basándose en la oferta que nos ofrece Dinahosting [24], un proveedor español afincado en Santiago de Compostela.

En nuestro caso, se necesitará un dominio y un hosting básico. Por tanto, necesitamos los siguientes elementos:

Descripción	Precio
Registro de dominio .com para VGA	14,00 €/año (primer año 7,00 €)
Hosting Linux Básico: • PHP • MariaDB/MySQL • 50 GB • 20 Cuentas de correo • Tráfico ilimitado	1 año: 54,00 € (primer año 27,00 €) 2 años: 104,40 € (52,20 €/año) 3 años: 151,20 € (50,40 €/año) 5 años: 216,00 € (43,20 €/año) 10 años: 378,00 € (37,80 €/año)

Tabla 10: Coste de mantenimiento de VGA

En este caso, se entiende que la mejor apuesta es realizar un registro por un año, con un coste total de 34 €. De este modo, podrá realizarse un estudio de la acogida de la herramienta a un coste razonable.

Al acercarse a la fecha de renovación será el punto de inflexión de si se sigue con la herramienta, renovando para uno o varios años, o si se decide desistir debido a que la acogida de la publicación del sitio no está teniendo la acogida esperada.

6 Pruebas

En este capítulo se detallan las pruebas que se han realizado durante el desarrollo para validar que el resultado de la herramienta es el esperado.

6.1 Pruebas de historias de usuario final

En este subcapítulo, se muestran las pruebas realizadas que verifican los criterios de aceptación a las historias de usuario.

6.1.1 Actualizar datos

En esta prueba se va a comprobar que la historia HU001 se realiza correctamente.

6.1.1.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el primer paso del algoritmo.

Pasos:

1 Actualizar los datos de entrada del algoritmo.
2 Pulsar en “Actualizar datos”
3 Ir al final del algoritmo.

6.1.1.2 Resultado esperado

La ejecución del algoritmo debe usar los nuevos datos de entrada. Por tanto, los datos de salida serán los obtenidos con la ejecución de los nuevos datos de entrada.

6.1.1.3 Resultado obtenido

Modificados los datos de entrada del algoritmo y tras la ejecución de este, los resultados obtenidos se corresponden con los datos de entrada introducidos.

6.1.1.4 Evidencias

Se utiliza el algoritmo de Dijkstra para las evidencias de esta historia de usuario.

Datos de entrada iniciales:

Datos de entrada del algoritmo

Circuito:

#	1	2	3	4	5
1	O	S	O	2	1
2	3	1	3	1	1
3	1	6	6	O	6
4	1	2	O	R	4
5	7	1	1	2	6

Circuito por el que se desplazara el robot (R) hasta la meta (S), con los obstaculos (O) y el coste de paso por una casilla indicado.

Figura 28: Datos de entrada del circuito

Camino resultante:

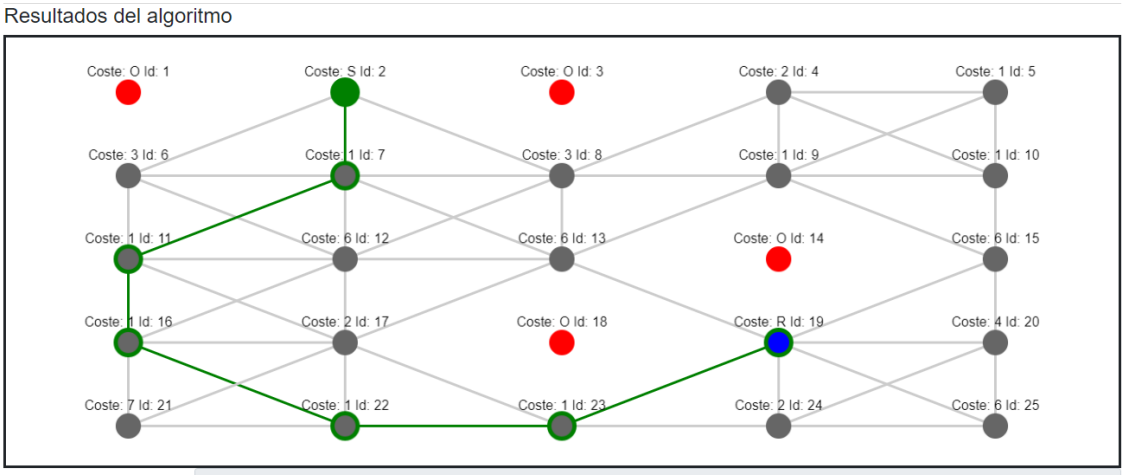


Figura 29: Camino resultante de la ejecución de Dijkstra

Modificamos los datos del algoritmo y pulsamos en “Actualizar datos”:

Figura 30: Nuevos datos de entrada para el algoritmo

Se reinicia el algoritmo:

Figura 31: Evidencia del reinicio del algoritmo

Nuevo resultado de la ejecución del algoritmo:

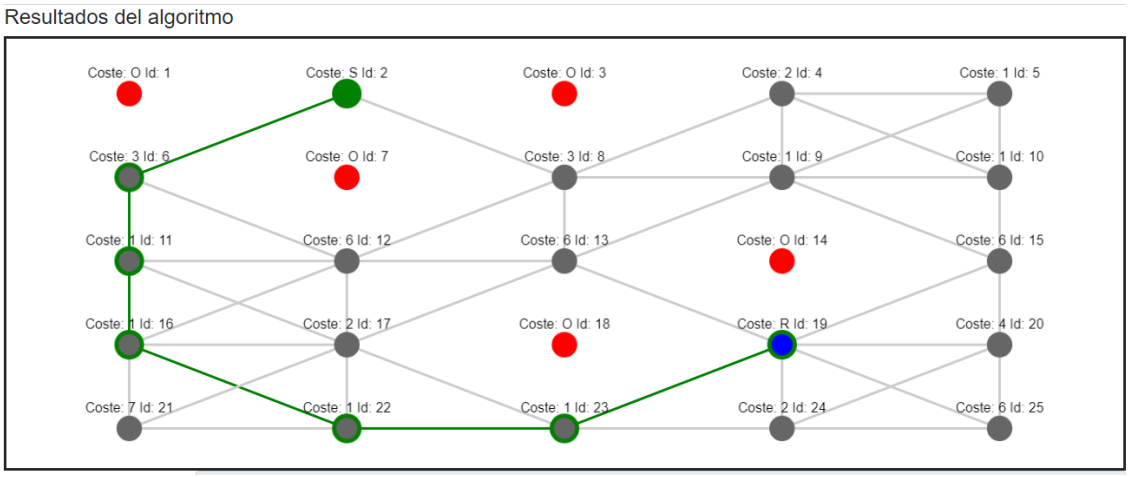


Figura 32: Nuevo camino del algoritmo con los nuevos datos de entrada

6.1.2 Ir al principio

En esta prueba se comprobará que la historia HU002 se realiza correctamente.

6.1.2.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el primer paso del algoritmo.

Pasos:

1 Pulsar en ir al principio ().

6.1.2.2 Resultado esperado

El algoritmo vuelve al primer paso colocando el cursor de código en la primera línea y reinicia todos los campos de la zona de resultados.

6.1.2.3 Resultado obtenido

Pulsado el botón de ir al principio, se establece el cursor de código en la primera línea y los campos de la zona de resultados se reinicia con los datos iniciales.

6.1.2.4 Evidencias

Situamos el algoritmo en un paso que no es el principio de este.

Figura 33: Algoritmo en un paso no inicial

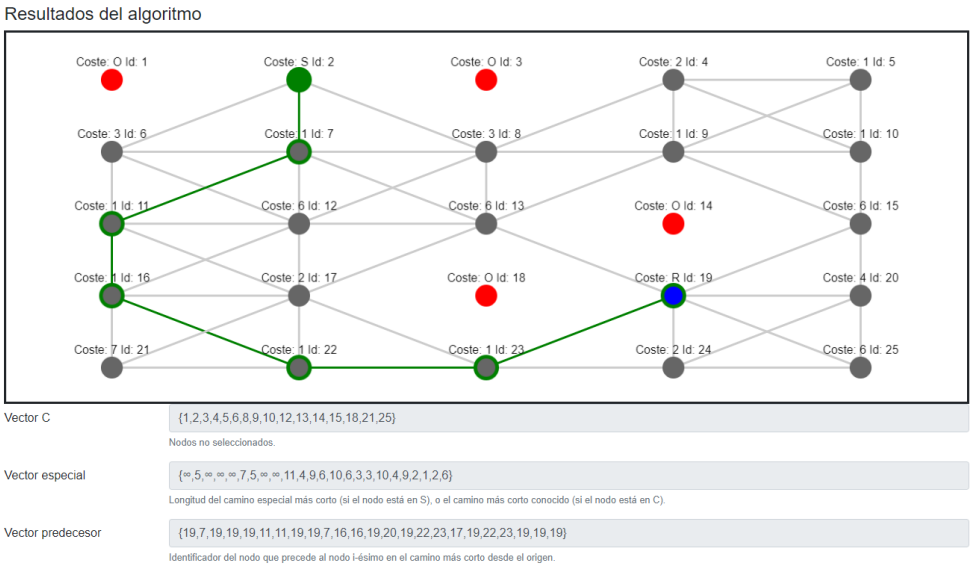


Figura 34: Estado de los campos de salida

Pulsamos en el botón de volver al principio. Se comprueba que el algoritmo volvió al primer paso y se reiniciaron los datos de los campos de salida a sus datos iniciales.

Figura 35: Situación del cursor de código en el inicio del algoritmo

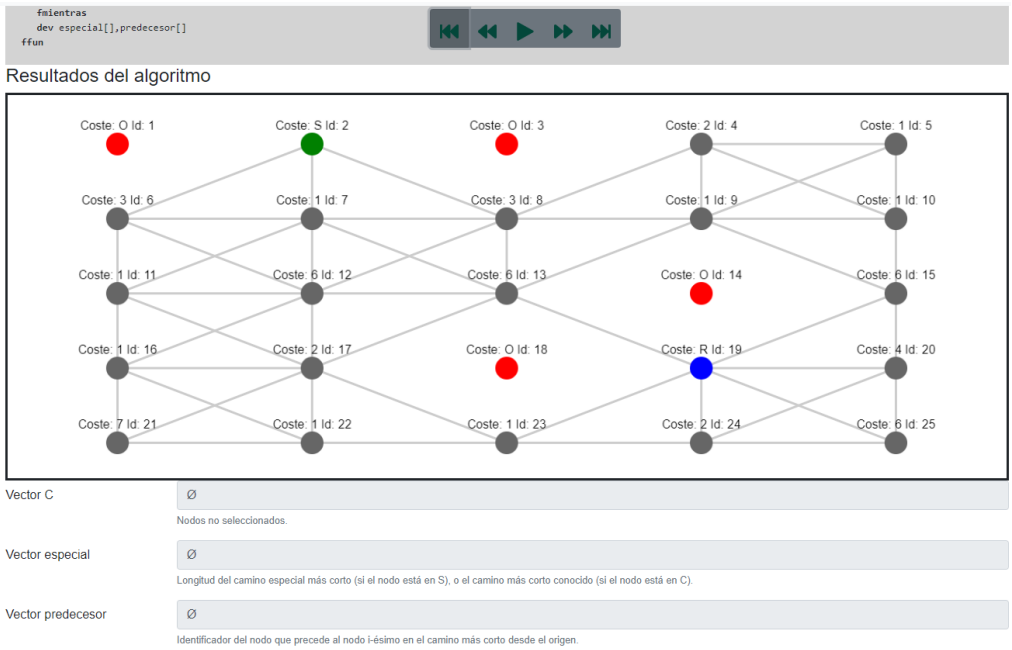


Figura 36: Datos de salida tras el reinicio del algoritmo

6.1.3 Retroceder

En esta prueba se comprobará que la historia HU003 se ejecuta correctamente.

6.1.3.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el primer paso del algoritmo.

Pasos:

1 Pulsar retroceder al paso anterior ().

6.1.3.2 Resultado esperado

El algoritmo vuelve al paso anterior colocando el cursor de código en la línea anterior y deshace los cambios en los datos de los campos de la zona de resultados.

6.1.3.3 Resultado obtenido

Pulsado el botón de ir al paso anterior, se establece el cursor de código en la línea anterior y se deshace los cambios en los datos de los campos de la zona de resultados.

6.1.3.4 Evidencias

Situamos el algoritmo en un paso que no es el principio de este.

Figura 37: Algoritmo en un paso no inicial

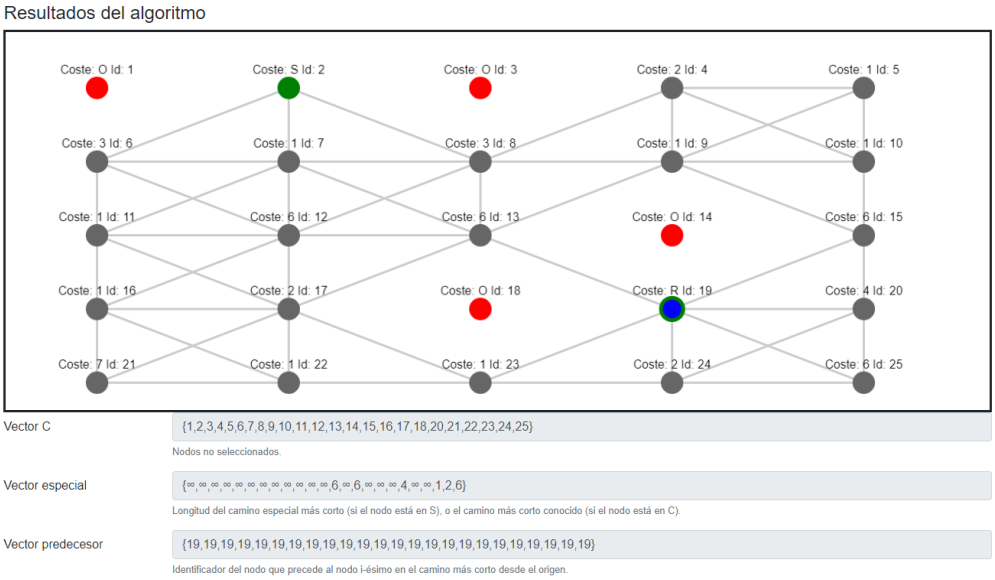


Figura 38: Estado de los campos de salida

Pulsamos en el botón de retroceder al paso anterior. Se comprueba que el algoritmo volvió al paso anterior y se deshicieron los cambios en los datos de los campos de salida.

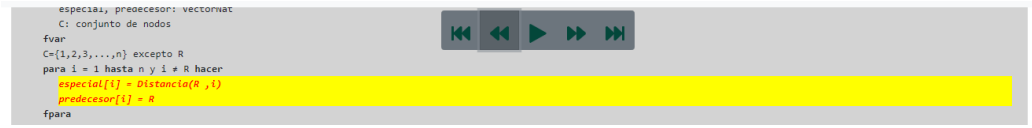


Figura 39: Situación el cursor de código al paso anterior

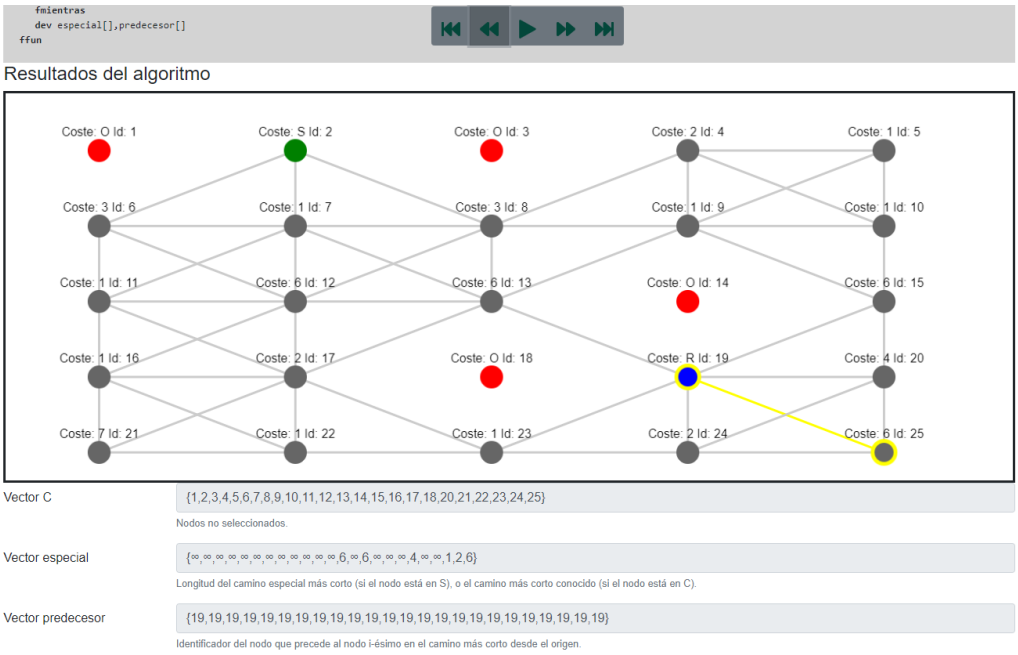


Figura 40: Datos de salida tras retroceder un paso del algoritmo

6.1.4 Reproducir

En esta prueba se comprobará que la historia HU004 se ejecuta correctamente.

6.1.4.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el último paso del algoritmo.
- La reproducción tiene que estar detenida.

Pasos:

1 Pulsar botón reproducir (▶).

6.1.4.2 Resultado esperado

El algoritmo avanza de un paso a otro automáticamente en el periodo de tiempo establecido en el sistema.

El botón de reproducir se modifica por el de pausar.

6.1.4.3 Resultado obtenido

Se reproduce paso a paso el algoritmo y se modifica el botón de reproducir por el de modificar.

6.1.4.4 Evidencias

Pulsamos el botón de reproducir:

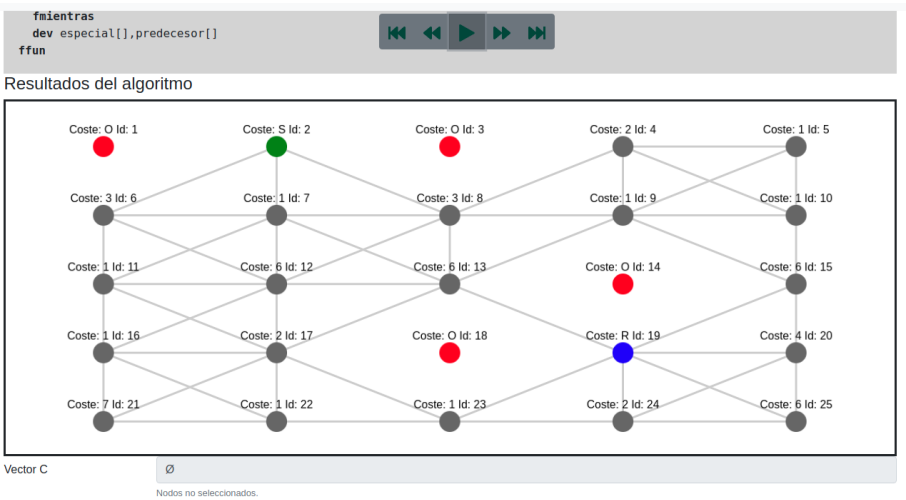


Figura 41: Algoritmo detenido en el primer paso

El algoritmo empieza a cambiar de paso periódicamente y se modifica el botón al de pausar el algoritmo:

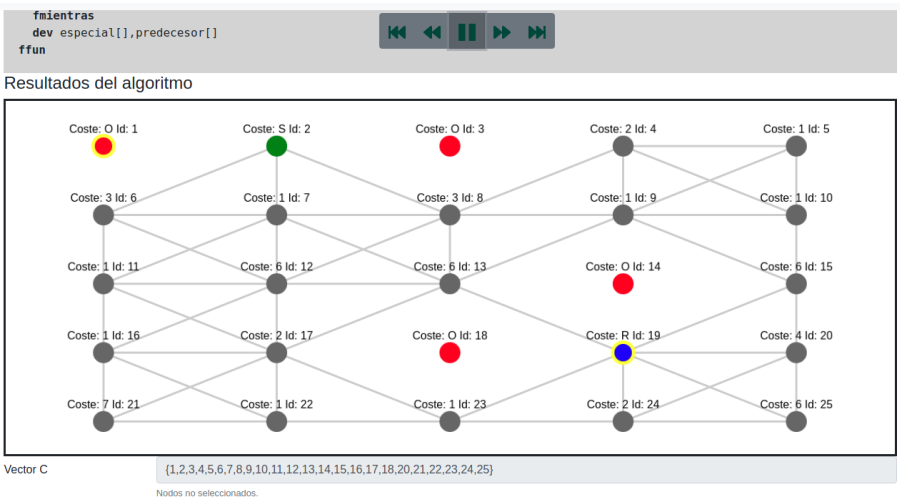


Figura 42: Algoritmo ejecutándose automáticamente

6.1.5 Pausar

En esta prueba se comprobará que la historia HU005 se ejecuta correctamente.

6.1.5.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el último paso del algoritmo.
- El algoritmo debe de estar ejecutándose.

Pasos:

1 Pulsar el botón de pausar ( ).

6.1.5.2 Resultado esperado

Se detiene el algoritmo en el paso actual.

Se modifica el botón de pausa por el de reproducir.

6.1.5.3 Resultado obtenido

El algoritmo se detiene y se modifica el botón de pausar por el de reproducir.

6.1.5.4 Evidencias

Pulsamos el botón de pausar:

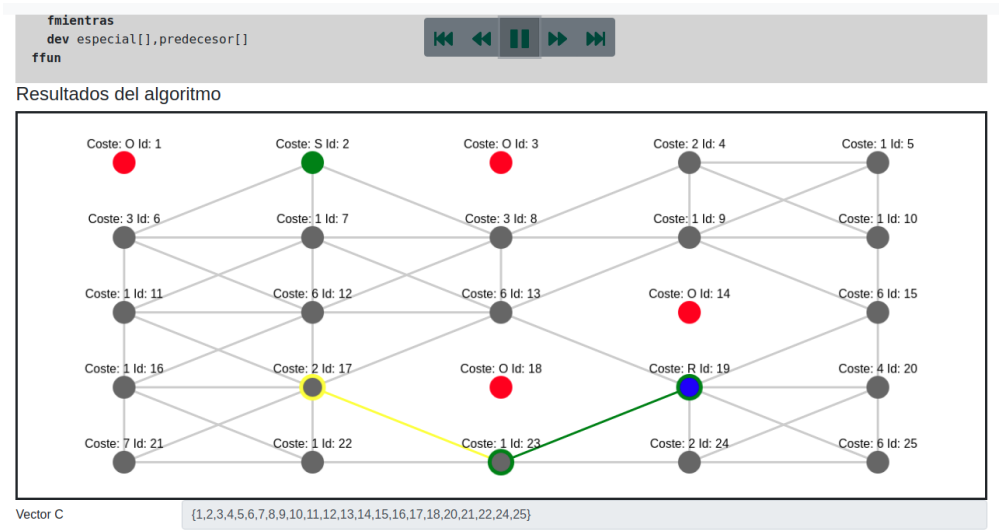


Figura 43: Algoritmo ejecutándose automáticamente

El algoritmo se detiene en el paso actual y se modifica el botón de pausar por el de reproducir.

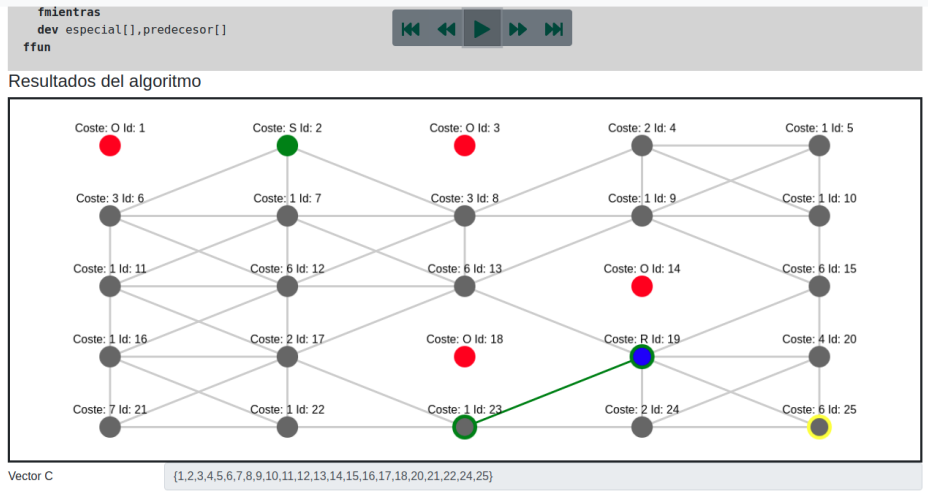


Figura 44: Algoritmo pausado satisfactoriamente

6.1.6 Avanzar

En esta prueba se comprobará que la historia HU006 se ejecuta correctamente.

6.1.6.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.

- No estar en el último paso del algoritmo.

Pasos:

1 Pulsar avanzar al paso siguiente ().

6.1.6.2 Resultado esperado

El algoritmo avanza al paso siguiente colocando el cursor de código en la línea siguiente y modifica los datos correspondientes a la ejecución del algoritmo de los campos de la zona de resultados.

6.1.6.3 Resultado obtenido

Pulsado el botón de ir al paso siguiente, se establece el cursor de código en la línea siguiente y se modifican los datos correspondientes de los campos de la zona de resultados.

6.1.6.4 Evidencias

Situamos el algoritmo en un paso que no es el final de este.

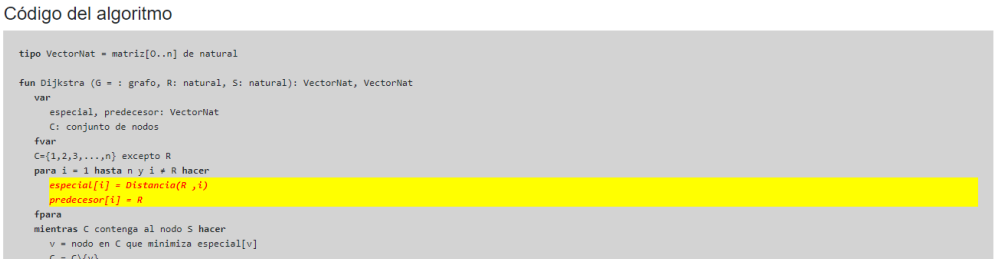


Figura 45: Algoritmo en un paso no final

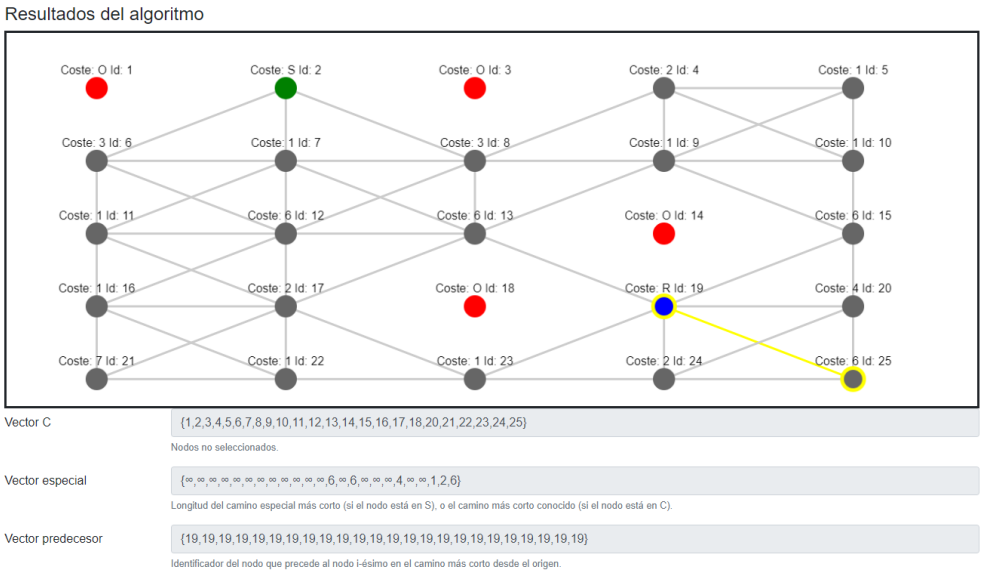


Figura 46: Estado de los campos de salida

Pulsamos en el botón de avanzar al paso siguiente. Se comprueba que el algoritmo avanzó al paso siguiente y se modificaron los datos correspondientes en los campos de salida.

Figura 47: Situación el cursor de código al paso posterior

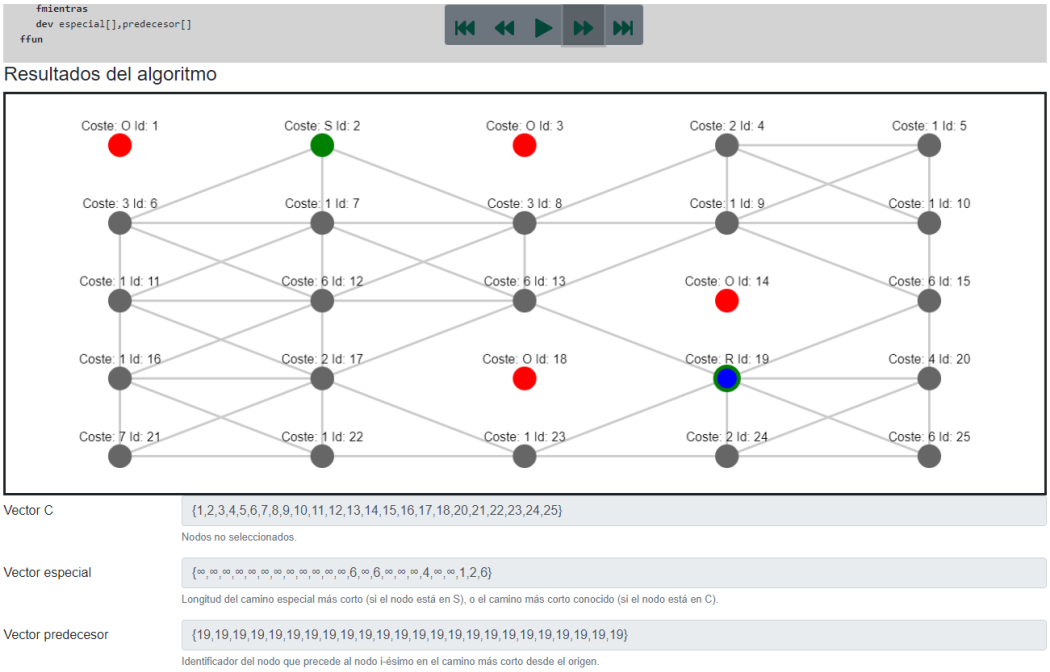


Figura 48: Datos de salida tras avanzar un paso del algoritmo

6.1.7 Ir al final

En esta prueba se comprobará que la historia HU007 se ejecuta correctamente.

6.1.7.1 Pasos de la prueba

Precondiciones:

- Estar en un algoritmo.
- No estar en el último paso del algoritmo.

Pasos:

1 Pulsar avanzar al último paso (⏭).

6.1.7.2 Resultado esperado

El algoritmo avanza al último paso colocando el cursor de código en la última línea y modifica los datos correspondientes a la ejecución del algoritmo de los campos de la zona de resultados.

6.1.7.3 Resultado obtenido

Pulsado el botón de ir al último paso, se establece el cursor de código en la última línea y se modifican los datos correspondientes de los campos de la zona de resultados.

6.1.7.4 Evidencias

Situamos el algoritmo en un paso que no es el final de este.

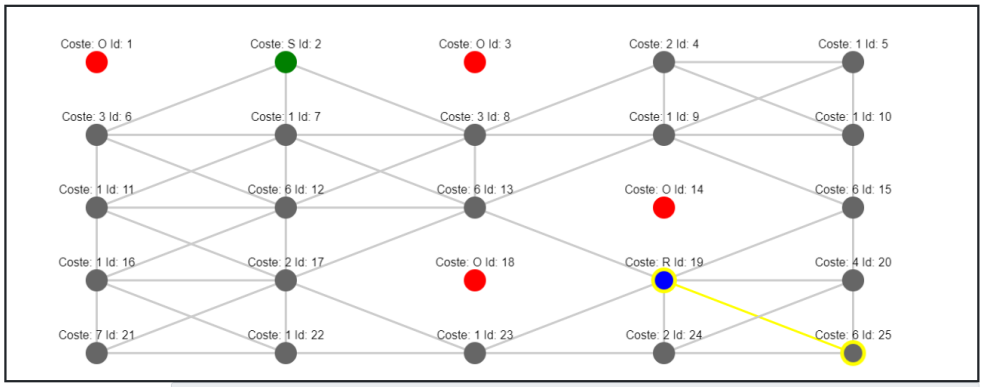
Código del algoritmo

```
tipo VectorNat = matriz[0..n] de natural

fun Dijkstra (G = : grafo, R: natural, S: natural): VectorNat, VectorNat
  var
    especial, predecesor: VectorNat
    C: conjunto de nodos
  fvar
  C={1,2,3,...,n} excepto R
  para i = 1 hasta n y i ≠ R hacer
    especial[i] = Distancia(R ,i)
    predecesor[i] = R
  fpara
  mientras C contenga al nodo S hacer
    v = nodo en C que minimiza especial[v]
```

Figura 49: Algoritmo en un paso no final

Resultados del algoritmo



Vector C

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,20,21,22,23,24,25}

Nodos no seleccionados.

Vector especial

{∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,6,∞,6,∞,∞,∞,4,∞,∞,1,2,6}

Longitud del camino especial más corto (si el nodo está en S), o el camino más corto conocido (si el nodo está en C).

Vector predecesor

{19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19}

Identificador del nodo que precede al nodo i-ésimo en el camino más corto desde el origen.

Figura 50: Estado de los campos de salida

Pulsamos en el botón de avanzar al último paso. Se comprueba que el algoritmo avanzó al último paso y se modificaron los datos correspondientes en los campos de salida.

Figura 51: Situación el cursor de código al último paso

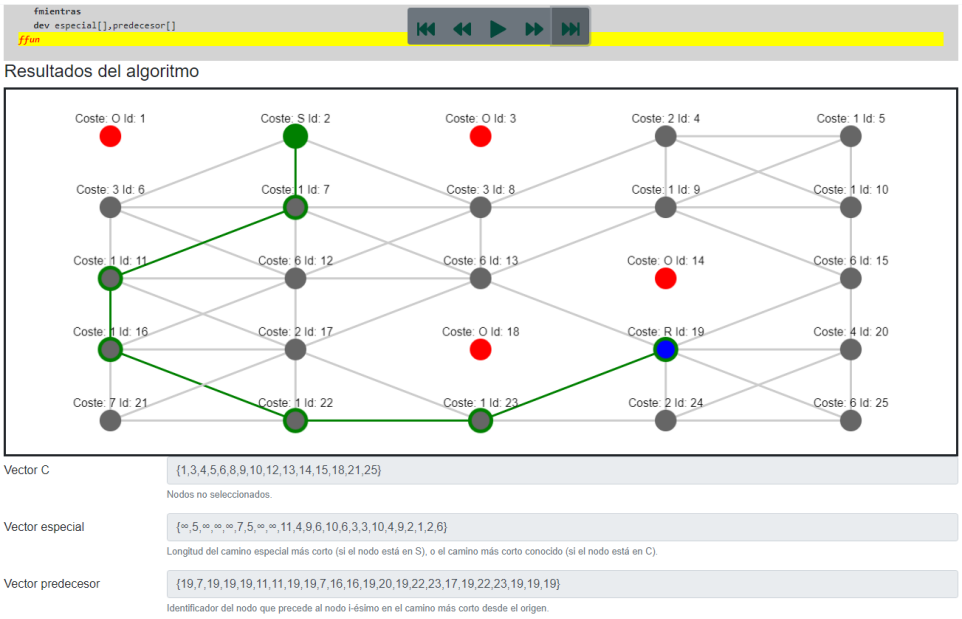


Figura 52: Datos de salida tras avanzar al último paso del algoritmo

6.2 Pruebas de historias de usuario desarrollador

6.2.1 Añadir nuevo control

En esta prueba se comprobará que la historia HU008 se realiza correctamente.

6.2.1.1 Resultado esperado

El sistema debe permitir añadir nuevos controles y utilidades que permitan aumentar las funcionalidades iniciales del sistema.

6.2.1.2 Resultado obtenido

Se permite modificar y añadir nuevos controles al sistema, permitiendo así aumentar las funcionalidades existentes.

6.2.1.3 Evidencias

Ver anexo *Tutorial distribución de un grafo en árbol*.

6.2.2 Añadir nuevo algoritmo

En esta prueba se comprobará que la historia HU009 se realiza correctamente.

6.2.2.1 Resultado esperado

Se desarrolla un marco de trabajo que permita añadir nuevos algoritmos al sistema.

6.2.2.2 Resultado obtenido

Se permite el desarrollo y mantenimiento de nuevos algoritmos dentro del sistema.

6.2.2.3 Evidencias

Ver anexo *Tutorial creación de algoritmo de ordenación Quicksort*.

6.3 Pruebas de algoritmos completos

En esta sección se realizan pruebas para diferentes algoritmos completos.

6.3.1 Mochila dinámica

En primer lugar, vamos a utilizar el problema de la mochila, resuelto mediante programación dinámica. Este problema consiste en rellenar una mochila, que tiene una capacidad determinada, con diferentes objetos, cada uno con un volumen y beneficio determinado. La solución al problema consiste en encontrar la relación de objetos que meter en la mochila, sin exceder su capacidad, maximizan el beneficio.

6.3.1.1 Definición del caso de prueba y resultado esperado

Para la ejecución de algoritmo de la mochila dinámica utilizaremos los siguientes objetos candidatos:

Volumen	Beneficio
1	2
5	14
7	15
4	10
3	5

Tabla 11: Objetos candidatos a entrar en la mochila

La mochila tendrá una capacidad de 10 unidades.

Tras la ejecución del algoritmo, el resultado esperado es que seleccione los elementos 1, 5 y 4 con un beneficio, que es máximo, de 26.

6.3.1.2 Resultado obtenido

Se ejecuta el algoritmo de la mochila dinámica con el resultado esperado.

6.3.1.3 Evidencias

Se muestran los datos de entrada del algoritmo:

Figura 53: Datos de entrada del algoritmo de la mochila dinámica

Se muestra el resultado de la ejecución, correspondiendo con el resultado esperado.

Resultados del algoritmo

Tabla de valores:

#	0	1	2	3	4	5	6	7	8	9	10
posición 0	0	0	0	0	0	0	0	0	0	0	0
$v_1 = 1\ b_1 = 2$	0	2	2	2	2	2	2	2	2	2	2
$v_2 = 5\ b_2 = 14$	0	2	2	2	2	14	16	16	16	16	16
$v_3 = 7\ b_3 = 15$	0	2	2	2	2	14	16	16	17	17	17
$v_4 = 4\ b_4 = 10$	0	2	2	2	10	14	16	16	17	24	26
$v_5 = 3\ b_5 = 5$	0	2	2	5	10	14	16	16	19	24	26

Tabla de datos usada por el algoritmo.

Objetos:

Índice objeto	1	2	3	4	5
Seleccionado	1	1	0	1	0

Vector de objetos que son elegidos para introducirse en la mochila.

Volumenes seleccionados: 1,5,4

Volumenes de los objetos que se introduciran en la mochila.

Beneficio: 26

Beneficio obtenido con los objetos selccionados, que es máximo.

Figura 54: Resultado de la ejecución del algoritmo de la mochila dinámica

6.3.2 Algoritmo de Dijkstra (Problema del robot)

En este problema, utilizaremos el algoritmo de Dijkstra para resolver el problema del robot desplazándose por un laberinto en forma de matriz bidimensional. Este problema consiste en un robot que debe desplazarse por un laberinto, con una serie de obstáculos. El robot tiene una energía inicial que va decreciendo a cada paso que da, según los pesos asignados a cada casilla.

La solución consiste en encontrar el camino entre la posición inicial del robot y la casilla de salida que suponga un menor gasto energético al robot. Para lo cual, se utiliza el algoritmo de Dijkstra, que nos permite encontrar los caminos mínimos desde un nodo inicial al resto de nodos de un grafo.

6.3.2.1 Definición del caso de prueba y resultado esperado

En la tabla que se muestra a continuación se indica la posición del robot (R), la meta (S), obstáculos (O) y los pesos de cada una de las casillas.

O	**S**	**O**	**2**	**1**
3	1	3	1	1
1	6	6	O	6
1	2	O	R	4

7	1	1	2	6

Tabla 12: Circuito de prueba del algoritmo de Dijkstra

Este circuito se corresponde con el siguiente grafo:

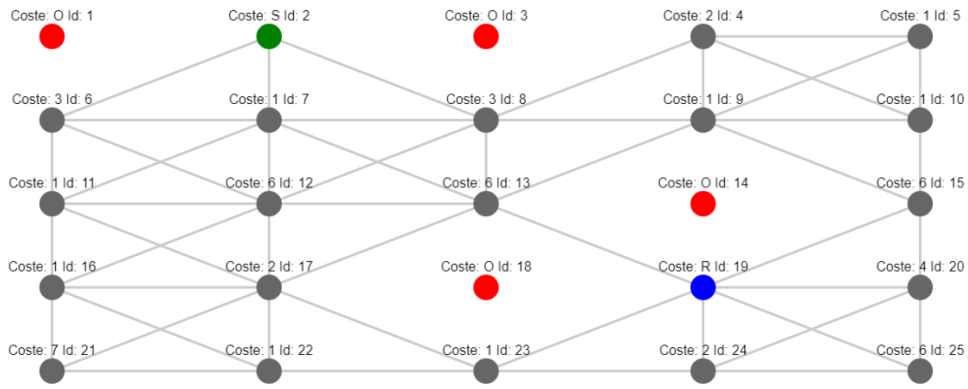


Figura 55: Grafo del circuito de prueba

De la ejecución del algoritmo se espera el resultado que se indica a continuación:

$$R[4,4],[5,3],[5,2],[4,1],[3,1],[2,2],S[1,2]$$

Figura 56: Camino resultante esperado

6.3.2.2 Resultado obtenido

Se ejecuta el algoritmo de Dijkstra y retorna el resultado esperado.

6.3.2.3 Evidencias

Se muestran los datos de entrada del algoritmo:

Datos de entrada del algoritmo

Circuito:

#	1	2	3	4	5
1	O	S	O	2	1
2	3	1	3	1	1
3	1	6	6	O	6
4	1	2	O	R	4
5	7	1	1	2	6

Circuito por el que se desplazara el robot (R) hasta la meta (S), con los obstaculos (O) y el coste de paso por una casilla indicado.

Figura 57: Datos de entrada de la prueba del algoritmo de Dijkstra

Se muestra el resultado de la ejecución, correspondiendo con el resultado esperado:

Resultados del algoritmo

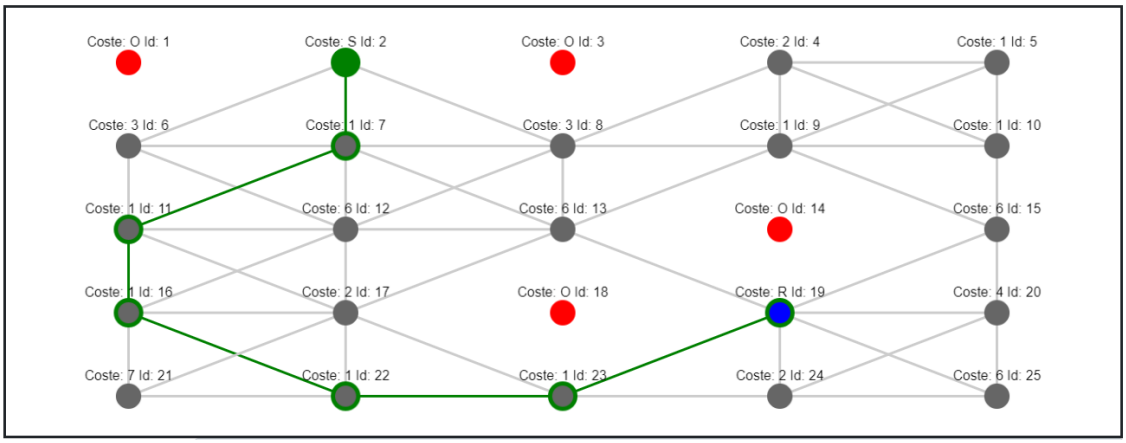


Figura 58: Resultado final de la prueba del algoritmo de Dijkstra

6.3.3 Algoritmo de ordenación Quicksort

Este algoritmo, desarrollado por Hoare, es considerado el mejor algoritmo de ordenación en tiempo medio. A cambio de su velocidad, plantea muchos problemas a los estudiantes a la hora de comprender su funcionamiento, por lo que consideramos que probar el funcionamiento de nuestro sistema con dicho algoritmo es algo imprescindible.

6.3.3.1 Definición del caso de prueba y resultado esperado

Para la ejecución de algoritmo de ordenación Quicksort utilizaremos los siguientes objetos a ordenar:

Objetos a ordenar
7,8,4,2,10,1,9,6,5,3

Tabla 13: Objetos a ordenar

Tras la ejecución del algoritmo, el resultado esperado es que los objetos queden ordenados de mayor a menor, es decir:

Objetos a ordenar
1,2,3,4,5,6,7,8,9,10

Tabla 14: Objetos ordenados

6.3.3.2 Resultado obtenido

Se ejecuta el algoritmo de la mochila dinámica con el resultado esperado.

6.3.3.3 Evidencias

Se muestran los datos de entrada del algoritmo:

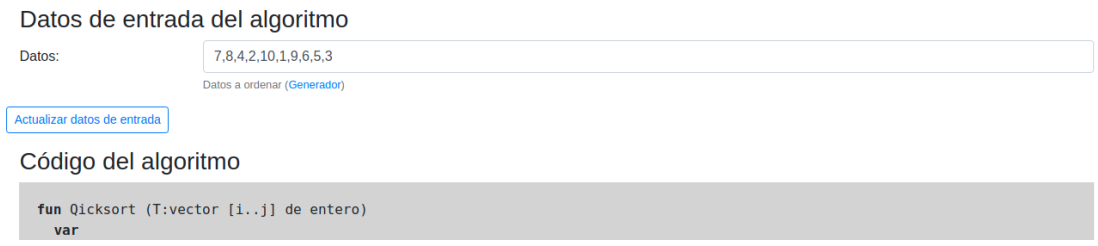


Figura 59: Datos de entrada del algoritmo Quicksort

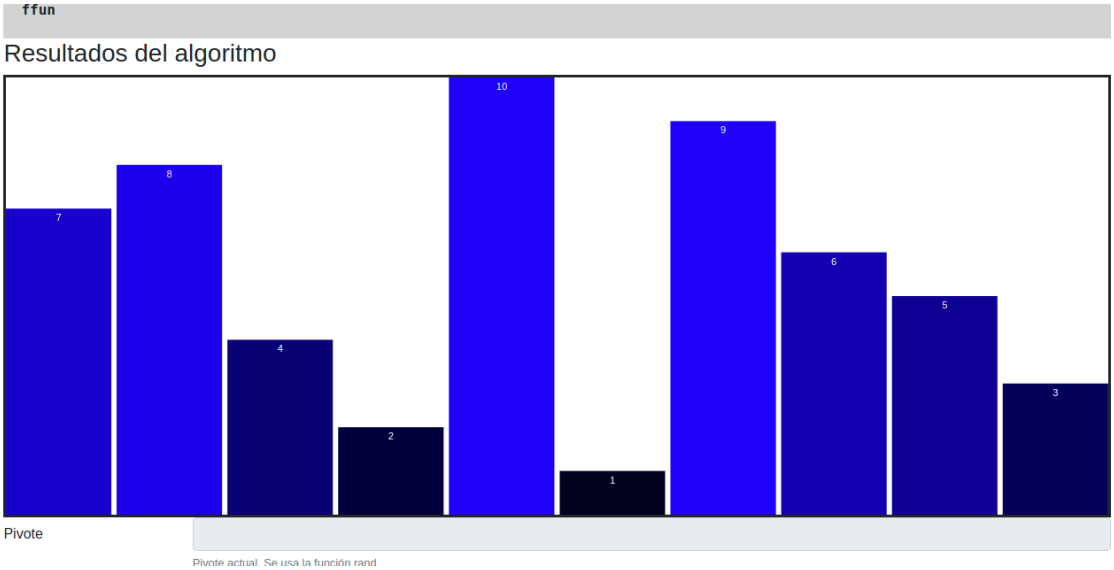


Figura 60: Gráfico del estado inicial del algoritmo Quicksort

Se muestra el resultado de la ejecución, correspondiendo con el resultado esperado.

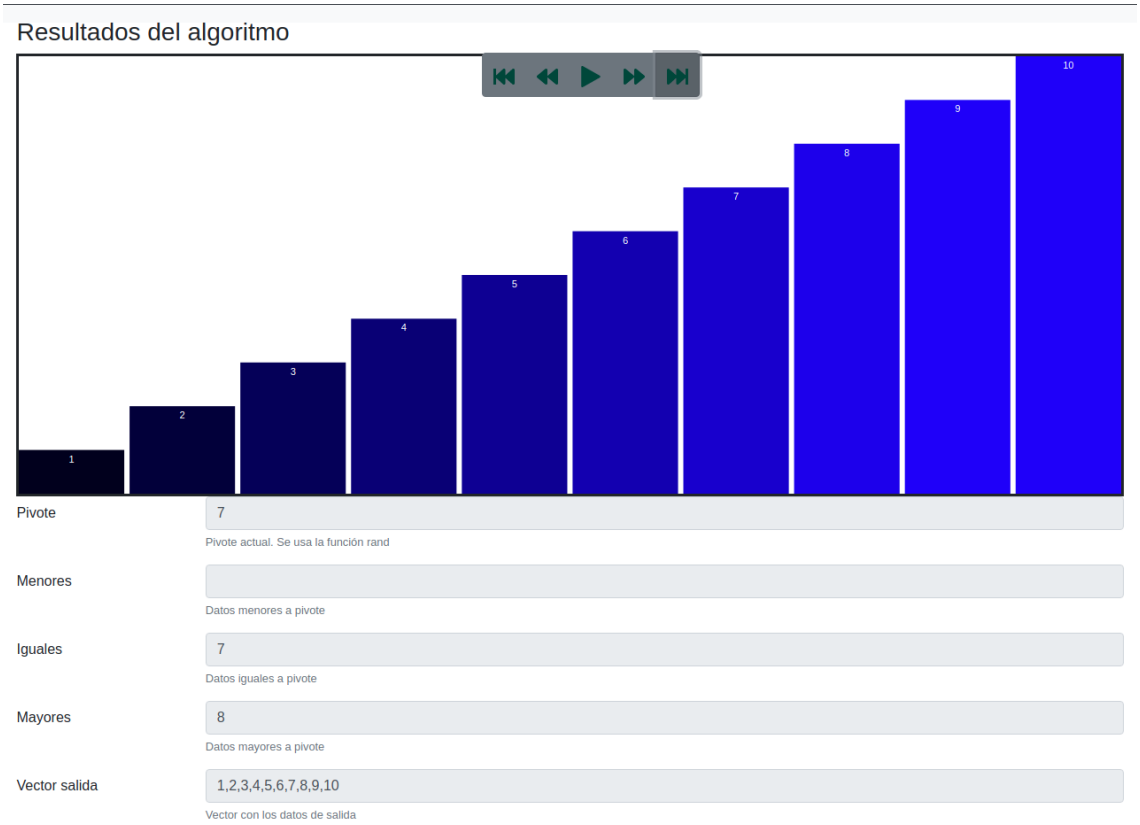


Figura 61: Datos de salida del algoritmo Quicksort tras su ejecución

7 Conclusiones y trabajos futuros

Este capítulo está dedicado a las conclusiones obtenidas tras el desarrollo del proyecto y a la descripción de posibles trabajos futuros que podrían mejorar la herramienta o ampliar su funcionalidad.

7.1 Conclusiones

Desde un primer momento, el proyecto ha sido un reto en cuanto a que el objetivo no estaba claro. Esto es debido a que la primera idea era crear un visor genérico de algoritmos sin ningún punto de partida. A partir de ese momento se fue refinando y enfocando lo que se pretendía hasta llegar a la situación actual del proyecto.

Aunque ya se partía con un conocimiento amplio de programación orientada a objetos y distintos frameworks de visualización de controles, se le dio una vuelta de tuerca al proyecto proponiendo que el algoritmo se ejecutase en JavaScript en el lado del cliente, sin un backend que se encargase de realizar los cálculos. Esto obligaba a una modificación del DOM del HTML basado en jQuery y JavaScript, lo cual resultaba un gran reto al no tener experiencia en este dominio.

Si hablamos de los objetivos marcados al comienzo del proyecto podemos que decir un sí con un depende. Se ha logrado realizar un entorno de ejecución de algoritmos ligero que permite la enseñanza y aprendizaje de algoritmos. Al intentar la máxima abstracción posible se ha logrado que el sistema pueda crecer con nuevos contenidos. Esto último es el depende comentado. Si esta herramienta no se sigue manteniendo y aumentado de contenido, no se habrá conseguido el objetivo de realizar una herramienta útil.

Con la realización de este proyecto, he podido aumentar mis capacidades de desarrollo de software que he aplicado tanto en este proyecto como en mi entorno laboral. El desarrollo de software no es sólo codificar, si no que es necesario una buena planificación y análisis de lo que se requiere para enforcar esfuerzos de codificación.

7.2 Trabajos futuros

Aunque la herramienta ya tiene un carácter funcional completo, podemos decir que esta versión es un punto de partida hacia un sistema que puede ayudar a los estudiantes y profesorado al aprendizaje y enseñanza de los algoritmos y estructuras internas.

A continuación se indican los trabajos futuros que pueden dar un valor añadido al presente proyecto.

7.2.1 Despliegue de VGA en un hosting

El despliegue de la herramienta en un hosting ayudaría considerablemente a la divulgación de la herramienta.

De esta manera, los docentes podrán hacer referencia a la url de los algoritmos directamente. Esto implica que los alumnos sólo tendrán que disponer de una conexión a internet y un navegador web actualizado.

Asimismo, esta acción ayudará a mantener en línea una versión actualizada de VGA, al poder configurar GitLab para que desencadene pipelines de publicación cuando se considere oportuno.

Finalmente, al ser este sistema desarrollado completamente pensando en una ejecución en cliente, el hosting seleccionado puede ser básico, reduciendo así el coste del mismo.

7.2.2 Ampliación de los algoritmos incluidos

Este sistema carecería de sentido si sólo se desarrollara para la ejecución de los tres algoritmos que tiene en la actualidad.

El sistema dispone de las herramientas necesarias para que se aumente el número de algoritmos y así ampliar el catálogo.

7.2.3 Ampliación de los controles existentes

Al igual que el sistema tiene capacidad para admitir nuevos algoritmos también lo hace para nuevos controles debido a su implementación siguiendo los principios de la orientación a objetos.

Nuevas estructuras y visualizaciones de las ya existentes pueden ser requeridas para correcta explicación de nuevos algoritmos. Pongamos por ejemplo la representación de un tablero de ajedrez para el problema de las ocho reinas. Esta estructura sería una especialización de la matriz bidimensional.

7.2.4 Adición de páginas complementarias con teoría

El proyecto se ha centrado en la elaboración de una API para la ayuda de la ejecución y visualización de las estructuras de los distintos algoritmos. No obstante, no nos podemos olvidar de que el sistema esta basado en un sitio web, por lo que se pueden añadir nuevas páginas en las cuales solo se incluyan datos teóricos.

Por ejemplo, pueden darse unas nociones sobre los árboles y su recorrido en anchura y en profundidad con enlaces a páginas donde se muestra el algoritmo que se usa para dicho fin.

8 Definiciones

Agile El desarrollo ágil de software envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

API Abreviatura de Application Programming Interface, interfaz de programación de aplicaciones en español. Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.[25]

BDD En la Ingeniería de Software, behavior-driven development o desarrollo guiado por el comportamiento, es un proceso de desarrollo de software que surgió a partir del desarrollo guiado por pruebas.

Código abierto (OpenSource) Se trata de software de código abierto y otros derechos que son publicados bajo una licencia de código abierto o son de dominio público.

Framework Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

JavaScript Lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Lenguaje de scripting para páginas web, y usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa. [26]

JS Diminutivo de JavaScript

TDD Desarrollo guiado por pruebas de software, o Test-driven development es una práctica de ingeniería de software que involucra otras dos prácticas: Escribir las pruebas primero y Refactorización. Para escribir las pruebas generalmente se utilizan las pruebas unitarias.

9 Bibliografía

1: A. Alliance, «What is Agile?», . [En línea]. Available: https://www.agilealliance.org/agile101/.

2: Wikipedia, «BDD», . [En línea]. Available: https://es.wikipedia.org/wiki/Desarrollo_guiado_por_comportamiento.

3: D. North, «What's in a Story?», . [En línea]. Available: https://dannorth.net/whats-in-a-story/.

4: Alexsoft, «Acceptance Criteria: Purposes, Formats, and Best Practices», . [En línea]. Available: https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/.

5: D. S. Halim, «Visualgo», . [En línea]. Available: https://visualgo.net/.

6: J. Park, «Algorithm Visualizer», . [En línea]. Available: https://algorithm-visualizer.org/.

7: A. Visualizer, «Algorithm Visualizer - Repositorio», . [En línea]. Available: https://github.com/algorithm-visualizer/algorithm-visualizer.

8: Facebook, «React», . [En línea]. Available: https://es.reactjs.org/.

9: O. Foundation, «Node.js», . [En línea]. Available: https://nodejs.org/es/.

10: Graphanalyzer, «Graph Online», . [En línea]. Available: https://graphonline.ru/es/.

11: Mark Allen Weiss, «Estructuras de datos en Java», 2013

12: Araujo, Lourdes, et al., «Programación y estructuras de datos avanzadas», 2011

13: Twitter, «BootStrap», . [En línea]. Available: https://getbootstrap.com/.

14: J. Resig, «jQuery», 2006. [En línea]. Available: https://jquery.com/.

15: I. f. S. Biology, «Cytoscape.js», . [En línea]. Available: https://js.cytoscape.org/.

16: J. H. V. O. y. l. c. Mike Bostock, «D3.js», 2011. [En línea]. Available: https://d3js.org/.

17: Microsoft, «Visual Studio Code», 2015. [En línea]. Available: https://code.visualstudio.com/.

18: Wikipedia, «Licencia MIT», 1968. [En línea]. Available: https://es.wikipedia.org/wiki/Licencia_MIT.

19: A. Systems, «Brackets», 2014. [En línea]. Available: http://brackets.io/.

20: ChangeVision, «Atash UML», . [En línea]. Available: https://astah.net/products/astah-uml/.

21: G. Inc., «GitLab», . [En línea]. Available: https://gitlab.com/.

22: Roger S. Pressman, «Ingeniería del software. Un enfoque práctico», 2010

23: QSM, «Function Point Languages Table», . [En línea]. Available: https://www.qsm.com/resources/function-point-languages-table.

24: Dinahosting, «Sitio web», . [En línea]. Available: https://es.dinahosting.com/.

25: Wikipedia, «Interfaz de programación de aplicaciones», . [En línea]. Available: https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones.

26: M. a. i. contributors, «MDN Web Docs», . [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

10 Anexos

10.1 Crear repositorio local

10.1.1 Introducción

Para empezar a trabajar en el desarrollo de nuevos algoritmos o de nuevas funcionalidades, es necesario crear un repositorio local donde descargarnos todo el código fuente y mantenerlo actualizado con el resto de los desarrolladores de la herramienta.

10.1.2 Cuenta en GitLab

Para poder realizar esta operación es necesario disponer de una cuenta en GitLab. Esta cuenta es gratuita y se puede hacer en cuestión de minutos en la página https://gitlab.com/.

También es necesario tener instalado git en nuestra computadora. Podemos descargar git desde aquí.

10.1.3 Crear la carpeta

Lo primero que debemos de hacer es crearnos una nueva carpeta donde alojaremos el código fuente de VGA.

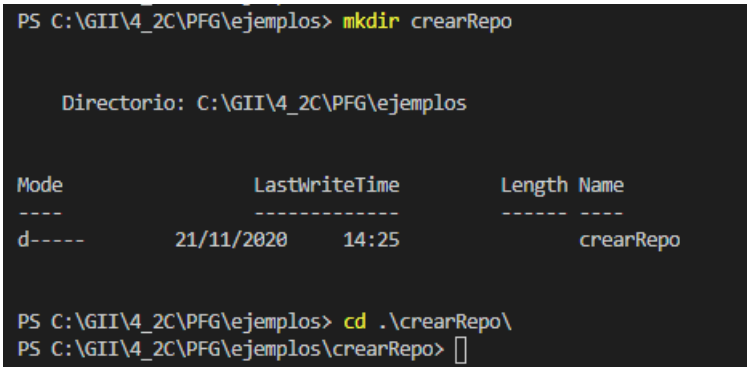


Figura 62: Creación de carpeta para el repositorio

Acto seguido, iniciaremos el repositorio ejecutando la siguiente orden:

Vía HTTP:

git clone https://gitlab.com/CarlosCaride/vga.git

Vía SSH:

git clone git@gitlab.com:CarlosCaride/vga.git

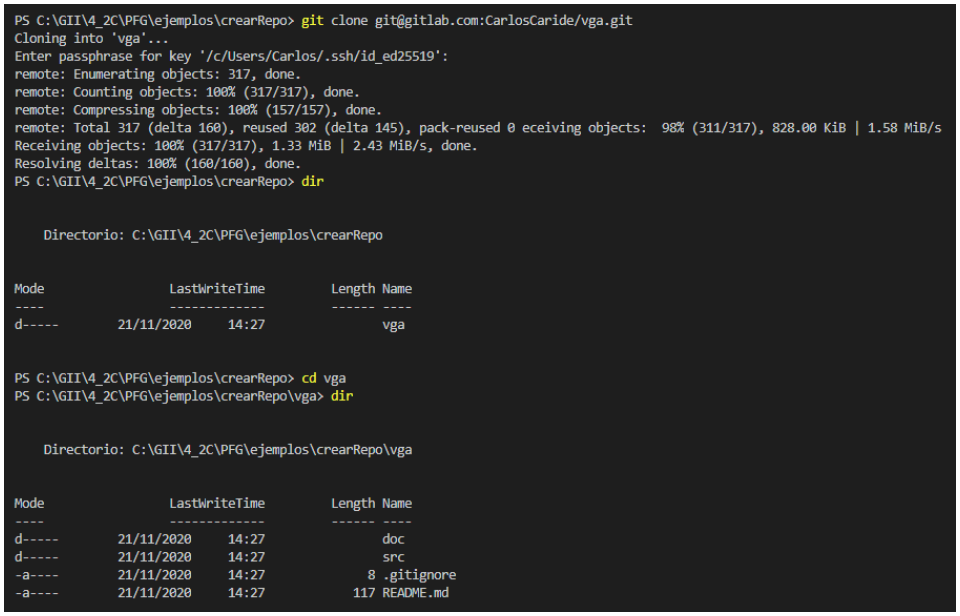


Figura 63: Resultado del clonado del repositorio

Pasados unos instantes, tendremos nuestro repositorio ya configurado para empezar a trabajar.

10.2 Tutorial distribución de un grafo en árbol

10.2.1 Introducción

En este tutorial se muestra cómo se puede aumentar las funcionalidades de VGA. Más concretamente, se vamos a añadir la distribución en árbol para un grafo, dado que no estaba implementado. Puede verse a continuación el resultado esperado.

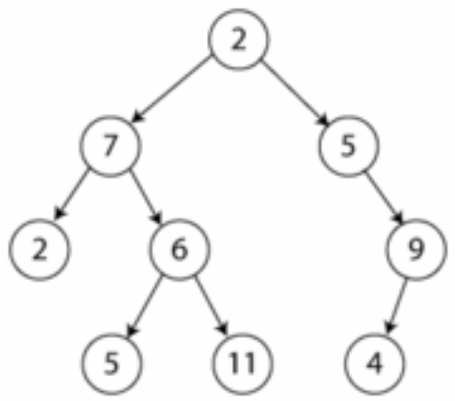


Figura 64: Distribución en forma de árbol

10.2.2 Crear la solicitud (incidencia)

Aunque es posible añadir nuevas funcionalidades sin necesidad de este apartado, lo ideal es que se realice mediante solicitudes (en GitLab incidencia). Los motivos que justifican esta decisión son los que siguen:

- Rastreabilidad de nuevos evolutivos para el proyecto.
- Asignación de tareas a desarrolladores.
- Comentarios sobre el nuevo evolutivo.
- Usuarios ajenos al desarrollo pueden reportar errores y sugerencias de mejora.

En los siguientes apartados, vamos a describir cómo crear la incidencia para su gestión.

10.2.2.1 Iniciar sesión en GitLab y acceder al repositorio del proyecto

Lo primero que se debe de hacer es iniciar sesión en GitLab, https://gitlab.com/.

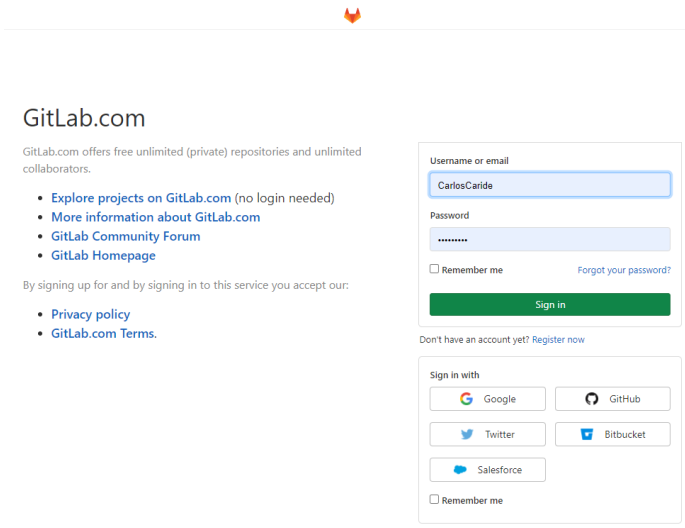


Figura 65: Pantalla de inicio de sesión en GitLab

Una vez iniciada la sesión, tenemos que acceder al repositorio del proyecto. Este se encuentra en la dirección https://gitlab.com/CarlosCaride/vga.

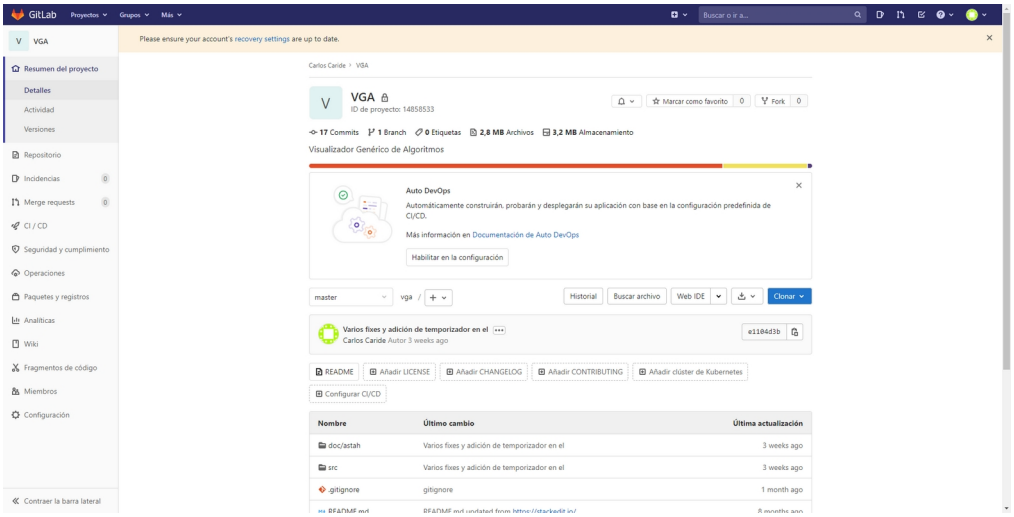


Figura 66: Pantalla principal del proyecto VGA

10.2.2.2 Creamos la incidencia

Accedemos a las incidencias tal y como muestra la imagen. En esta pantalla se verán las incidencias que están abiertas en estos momentos, pudiendo acceder a ellas para ver su estado y comentarios. También se pueden consultar las incidencias cerradas.

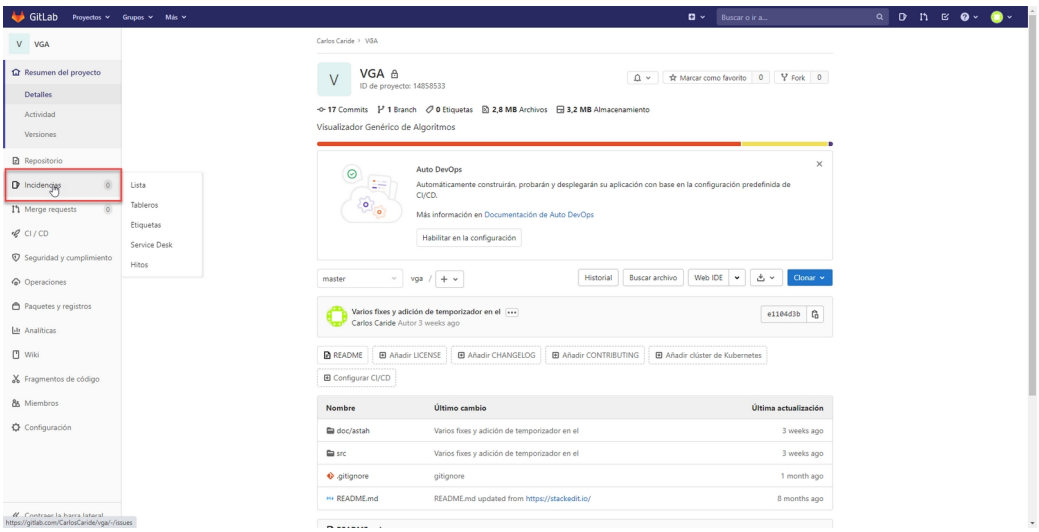


Figura 67: Acceder a las incidencias

Para crear una nueva incidencia, solo es necesario pulsar sobre el botón *Nueva incidencia*. Esto nos llevará al formulario de creación de una nueva incidencia.

Figura 68: Botón crear nueva incidencia

Para este ejemplo, los datos que debemos rellenar son los siguientes:

- **Título:** Grafo. Disposición en árbol
- **Type:** Issue (Incidencia)
- **Description:** “Hola, Me gustaría disponer de la distribución en árbol para los grafos. Gracias”
- **Labels:** api, funcionalidad, grafo.

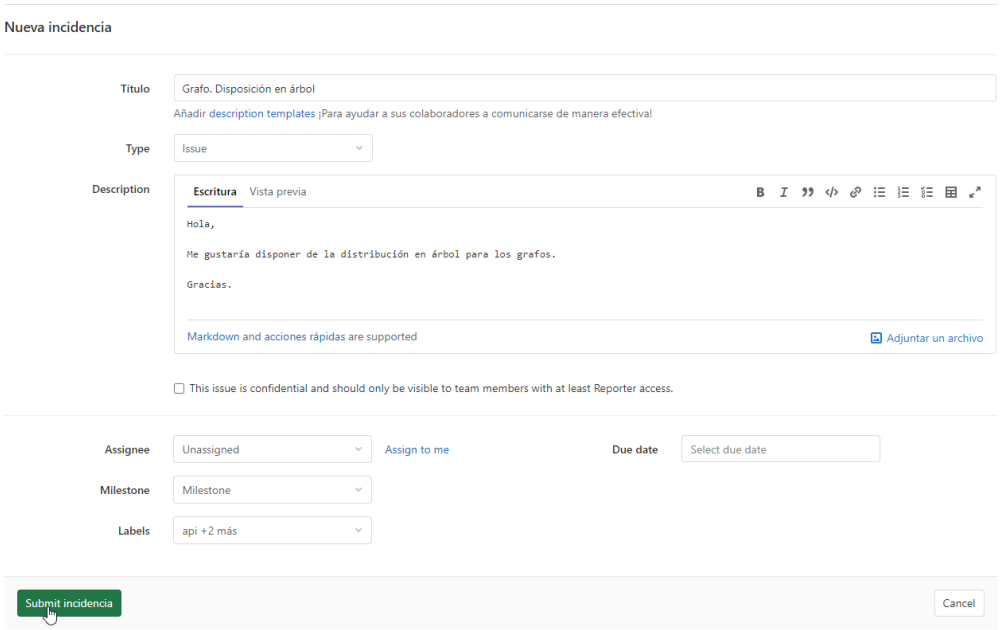


Figura 69: Formulario de creación de incidencia

Finalmente, pulsando en *Submit incidencia* la crearemos para su gestión. Como puede observarse, se pueden modificar otros datos, como la asignación, tiempos de entrega e hito.

10.2.3 Crear la rama para esta petición

Lo primero de todo, hay que explicar que es una rama en un repositorio Git. Git almacena los datos en forma de instantáneas. En cada confirmación, Git almacena esos datos y avanza una posición. Git crea por defecto una rama denominada *master*.

Cuando se desarrolla una nueva funcionalidad, sobre todo entre varios desarrolladores, puede ser que se realicen varios *comits* antes de finalizar su desarrollo. Sin embargo, tenemos que dejar el proyecto listo para tener que desplegarlo cuando sea necesario. Como se puede entender, añadir nuevas funcionalidades puede hacer que el proyecto no esté estable del todo. Ir añadiendo nuevos *comits* a *master* parece que no es buena idea.

Otro caso que se nos puede dar es que se estén desarrollando dos o más funcionalidades en paralelo, por lo que la complejidad aumenta.

Todos estos problemas los podemos solucionar con lo que se denomina *branching*, esto es, crear ramas únicas a partir de *master* para desarrollar la funcionalidad. Cuando la funcionalidad esté finalizada y estable, se unirá de nuevo a *master*. Con ello, se logra el aislamiento de la parte de desarrollo con la de producción. En la siguiente imagen puede verse el concepto de las ramas en Git.

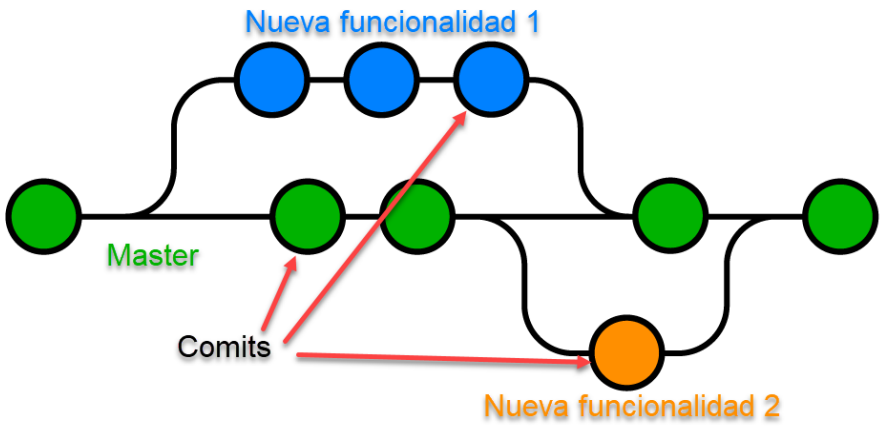


Figura 70: Modelo de branching para VGA

Aunque existen multitud de tipos de modelos, entendemos que este modelo es más que suficiente para el proyecto actual.

Volviendo a nuestro caso, dentro de la incidencia podemos crear la rama, que en este caso le llamaremos *rama-grafo-arbol*.

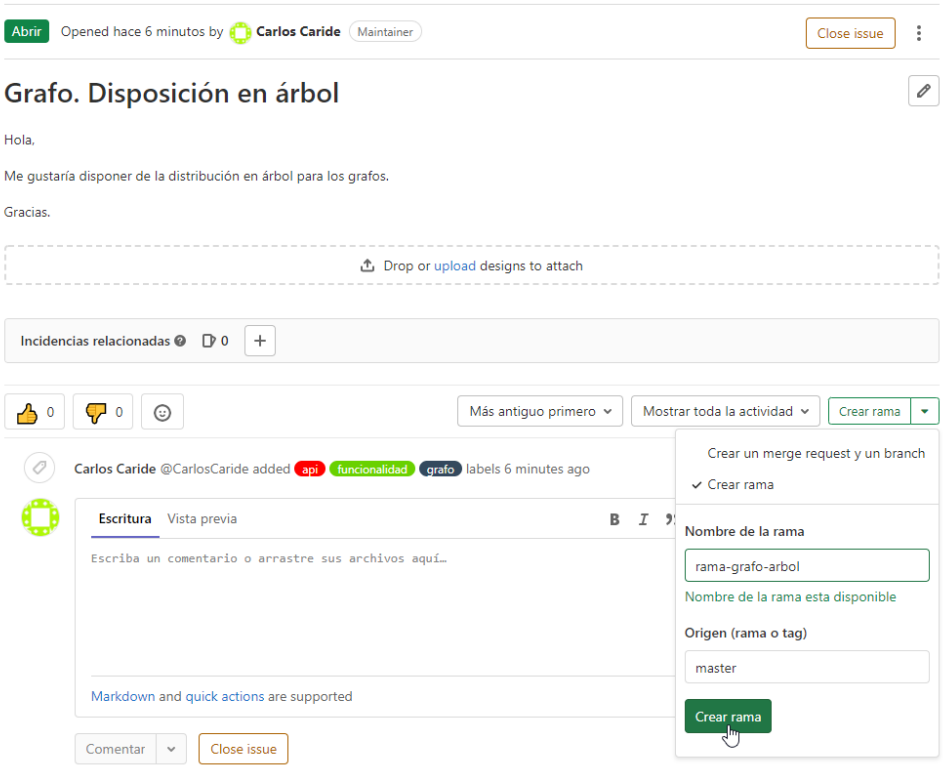


Figura 71: Crear rama

Una vez creada la rama, debería existir en la lista de ramas y ya podremos ponernos a desarrollar nuestra nueva funcionalidad.

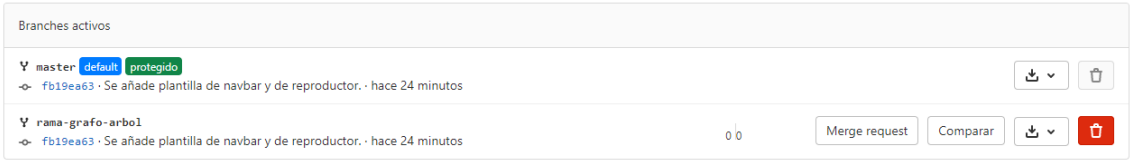


Figura 72: Listado de ramas del proyecto

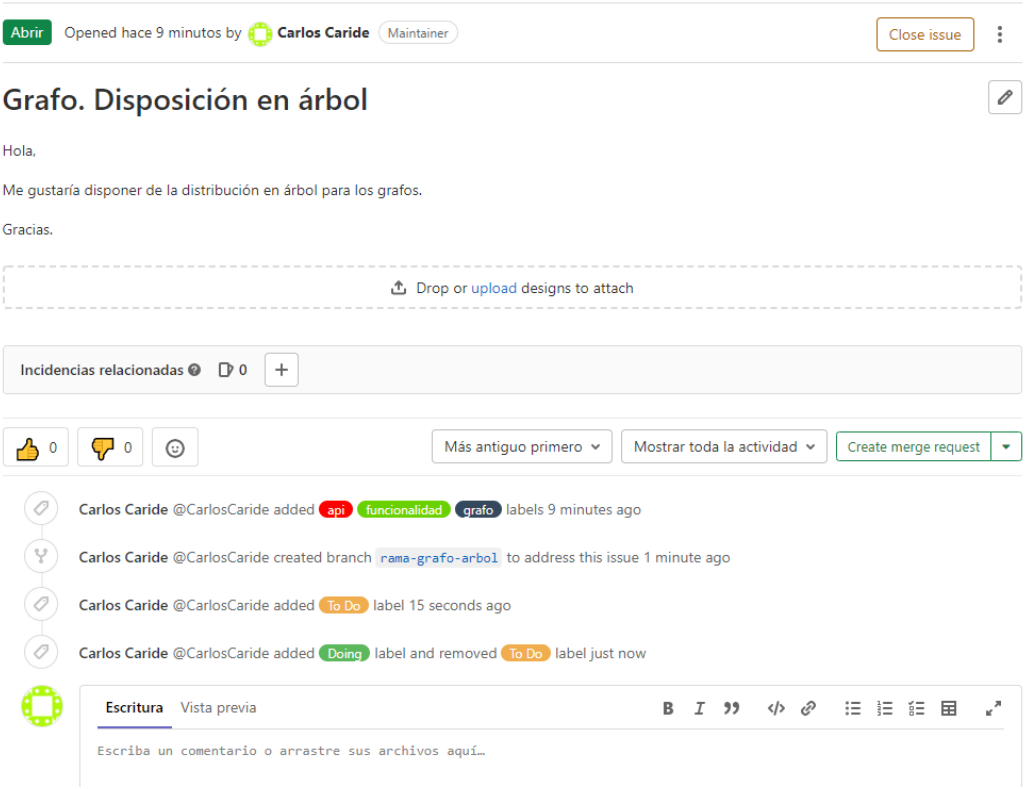


Figura 73: Historial de la incidencia

10.2.4 Traerse la rama para local

Antes de desarrollar la nueva funcionalidad, debemos de traernos la rama que hemos creado en GitLab a nuestro repositorio local. Si no tienes creado el repositorio, consulta el anexo de creación el repositorio.

Para traernos las ramas, lo primero que tenemos que hacer es actualizar las referencias al repositorio remoto. Eso se hace con el comanto *git fetch -p*. Podemos comprobar que la operación resultó correcta con el comando *git branch -a*, que nos mostrará las ramas locales y remotas.

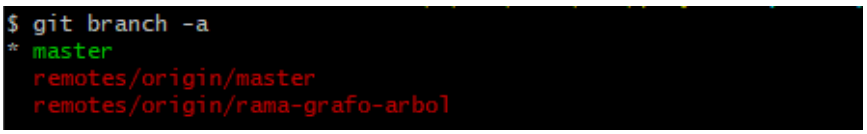


Figura 74: Listar ramas locales y remotas

Como podemos ver, en este caso estamos en la rama master local, pero necesitamos crear la nueva rama y cambiarnos a ella. Para ello, solo es necesario ejecutar el comando que se muestra en la siguiente imagen:

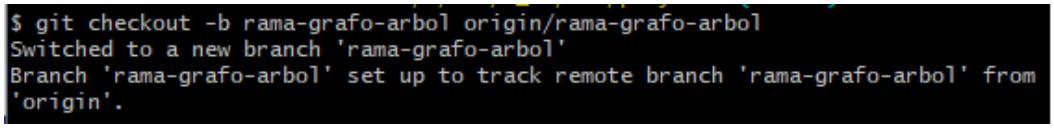


Figura 75: Creación de nueva rama local

Si volvemos a comprobar las ramas, podemos comprobar que hemos creado la nueva rama, y que apuntamos a ella. De este modo, salvaguardamos la rama master de este desarrollo hasta que no esté finalizado.

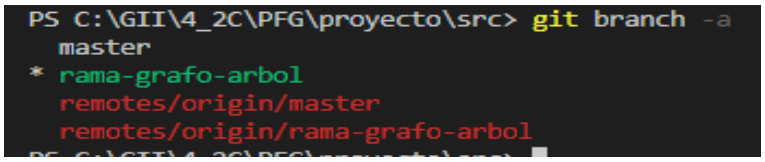


Figura 76: Comprobación de la creación de la nueva rama

Ahora, sí que estamos en situación de empezar con el desarrollo de la nueva función de VGA.

10.2.5 Desarrollo de la nueva distribución

Para este desarrollo, vamos a buscar información en la propia página del framework Cytoscape.js. En la documentación, se observa que dispone de varios tipos de layout, siendo uno de ellos el que nos interesa para VGA.

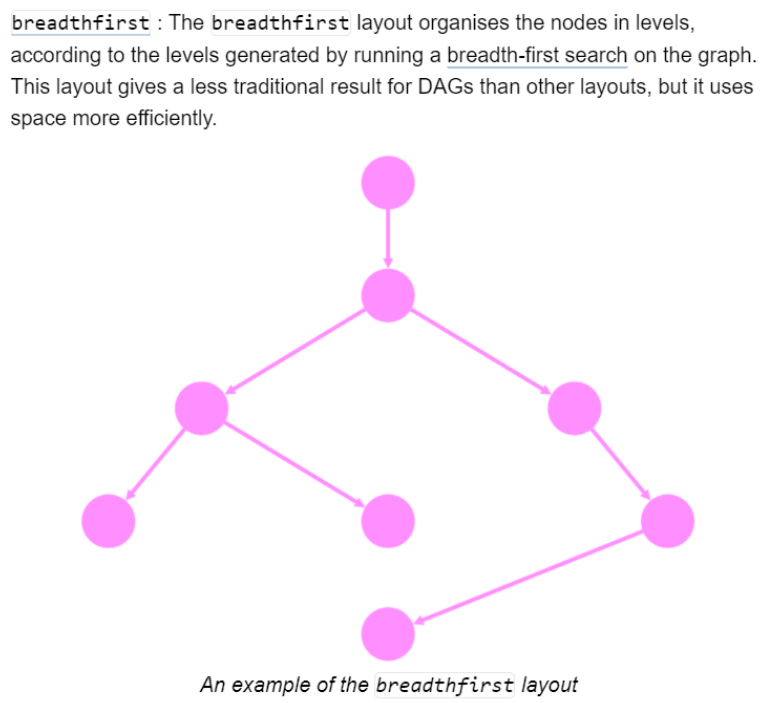


Figura 77: Distribución en forma de árbol en la documentación de Cytoescape.js

Asimismo, se incluye código de ejemplo de cómo configurar el grafo. Por tanto, procedemos a añadir una nueva clase derivada de *Layout*, dentro del archivo *vga.graph.js*:

```
class TreeLayout extends Layout {
    constructor() {
        super('breadthfirst');
    }

    getLayout(graph) {
        return graph.elements().layout({
            name: this.getLayoutType(),

            directed: true,

            nodeSep: undefined,
            edgeSep: undefined,
            rankSep: undefined,
            rankDir: undefined,
            ranker: undefined,
            minLen: function (edge) { return 1; },
            edgeWeight: function (edge) { return 1; },

            fit: true,
            padding: 30,
            spacingFactor: undefined,
            nodeDimensionsIncludeLabels: false,
            animate: false,
            animateFilter: function (node, i) { return true; },
```

```
            animationDuration: 500,
            animationEasing: undefined,
            boundingBox: undefined,
            transform: function (node, pos) { return pos; },
            ready: function () { },
            stop: function () { }
        });
    }
}
```

Una vez desarrollada esa parte, vamos a hacer uso de la plantilla para probar que la distribución en forma de árbol es correcta. Para ello, añadiremos un nuevo grafo en la zona de resultados con los datos hardcodeados:

```
function inicializarZonaResultados() {
    var arbol = new Graph("arbol", "Test arbol", "Test de un arbol");

    OutputData.addControl(arbol);

    var estilosGrafo = [
        {
            selector: 'node',
            style: {
                'background-color': '#666',
                'label': 'data(label)'
              }
        },
        {
            selector: 'edge',
            style: {
                'curve-style': 'bezier',
                'width': 6,
                'target-arrow-shape': 'triangle',
                'line-color': '#ffaaaa',
                'target-arrow-color': '#ffaaaa'
            }
        }
    ];
    arbol.initGraph(estilosGrafo);
    arbol.addNode("cat", "cat");
    arbol.addNode("bird", "bird");
    arbol.addNode("gusano", "gusano");
    arbol.addNode("ladybug", "ladybug");
    arbol.addNode("aphid", "aphid");
    arbol.addNode("rose", "rose");
    arbol.addNode("grasshopper", "grasshopper");
    arbol.addNode("plant", "plant");
    arbol.addNode("wheat", "wheat");

    arbol.addEdge("cat-bird", "cat", "bird");
    arbol.addEdge("bird-ladybug", "bird", "ladybug");
    arbol.addEdge("bird-gusano", "bird", "gusano");
    arbol.addEdge("bird-grasshopper", "bird", "grasshopper");
    arbol.addEdge("grasshopper-plant", "grasshopper", "plant");
    arbol.addEdge("grasshopper-wheat", "grasshopper", "wheat");
    arbol.addEdge("ladybug-aphid", "ladybug", "aphid");
    arbol.addEdge("aphid-rose", "aphid", "rose");

    arbol.setLayout(new TreeLayout());
}
```

El resultado final puede comprobarse en la siguiente ilustración, dando por correcto el resultado obtenido.

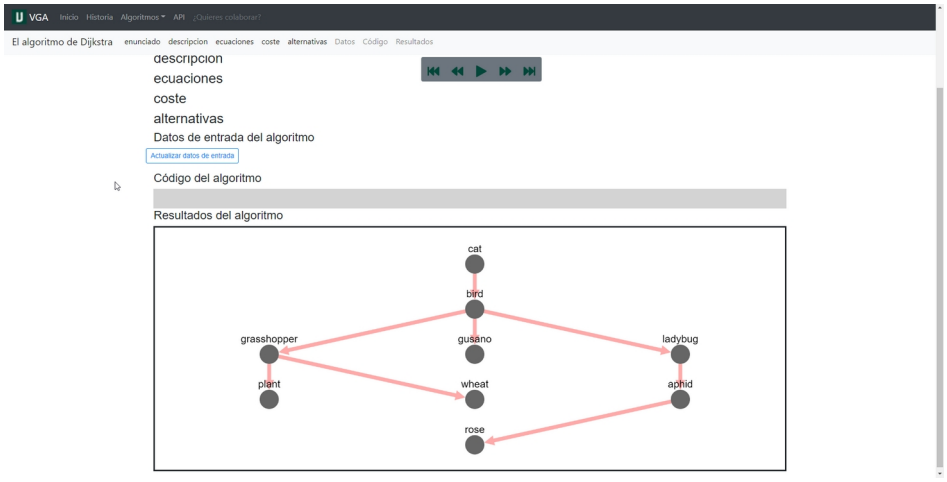


Figura 78: Resultado de la implementación de la distribución en árbol

10.2.6 Envío de cambios a GitLab

Lo primero que tenemos que hacer es añadir el archivo *vga.graph.js* al conjunto de cambios que vamos a enviar a GitLab. Después, debemos realizar un commit, es decir, aceptar los cambios en el repositorio local. En la siguiente imagen se muestran los comandos a ejecutar.

```
PS C:\GII\4_2C\PFG\proyecto\src> git st
## rama-grafo-arbol...origin/rama-grafo-arbol
 M algoritmos/plantilla.js
 M algoritmos/voraces/dijkstra.html
 M js/vga-core/vga.graph.js
PS C:\GII\4_2C\PFG\proyecto\src> git add .\js\vga-core\vga.graph.js
PS C:\GII\4_2C\PFG\proyecto\src> git st
## rama-grafo-arbol...origin/rama-grafo-arbol
 M algoritmos/plantilla.html
 M algoritmos/plantilla.js
 M algoritmos/voraces/dijkstra.html
M  js/vga-core/vga.graph.js
PS C:\GII\4_2C\PFG\proyecto\src> git commit -m "Se añade la distribución en forma de árbol (TreeLayout)"
[rama-grafo-arbol 965d16a] Se añade la distribución en forma de árbol (TreeLayout)
 1 file changed, 38 insertions(+), 2 deletions(-)
```

Figura 79: Commit al repositorio local de los cambios realizados

No obstante, y como hemos indicado, los cambios están guardados a nivel local, al repositorio de nuestro ordenador. Para enviar los cambios a GitLab es

necesario ejecutar el comando *git push*, que enviará todos los cambios que hemos realizado en nuestro repositorio local al remoto.

```
PS C:\GII\4_2C\PFG\proyecto\src> git push
Enter passphrase for key '/c/Users/Carlos/.ssh/id_ed25519':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.38 KiB | 472.00 KiB/s, done.
Total 6 (delta 5), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for rama-grafo-arbol, visit:
remote:   https://gitlab.com/CarlosCaride/vga/-/merge_requests/new?merge_request%5Bsource_branch%5D=rama-grafo-arbol
remote:
To gitlab.com:CarlosCaride/vga.git
   fb19ea6..965d16a  rama-grafo-arbol -> rama-grafo-arbol
```

Figura 80: Realizando git push

Finalmente, procedemos a eliminar la rama en la que hemos estado trabajando. Para ellos nos movemos a master con el comando git checkout master y eliminamos la rama del desarrollo con git Branch -d rama-grafo-arbol.

```
PS C:\GII\4_2C\PFG\proyecto\src> git checkout master
Switched to branch 'master'
```

Figura 81: Cambio a la rama master

```
PS C:\GII\4_2C\PFG\proyecto\src> git branch
* master
  rama-grafo-arbol
PS C:\GII\4_2C\PFG\proyecto\src> git branch -d rama-grafo-arbol
Deleted branch rama-grafo-arbol (was 965d16a).
PS C:\GII\4_2C\PFG\proyecto\src>
```

Figura 82: Eliminación de la rama rama-grafo-arbol

10.2.7 Solicitamos el merge request

Una vez realizado los cambios y enviados a GitLab, el desarrollador debe solicitar que la rama que se ha creado para el desarrollo se una a la rama principal. Para ello, se debe de crear una solicitud de fusión. En ella indicamos que es lo que se ha realizado. Opcionalmente, marcaremos que se elimine la rama del repositorio, ya que no tiene sentido mantenerla.

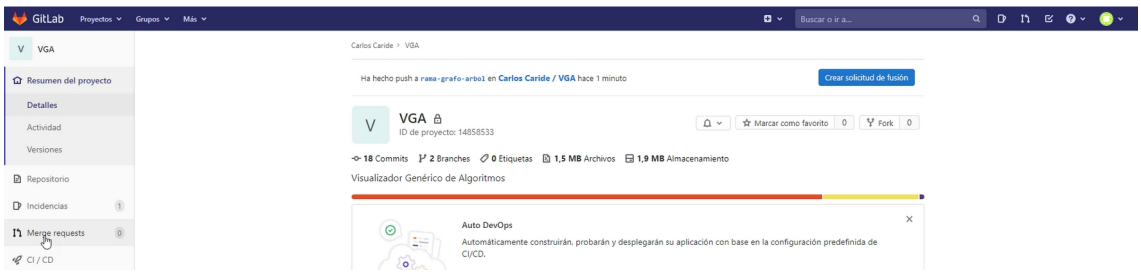


Figura 83: Acceso a los merge request

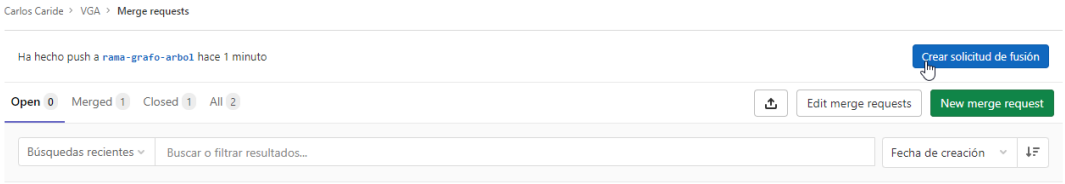


Figura 84: Acceso al formulario de fusión para el cambio realizado

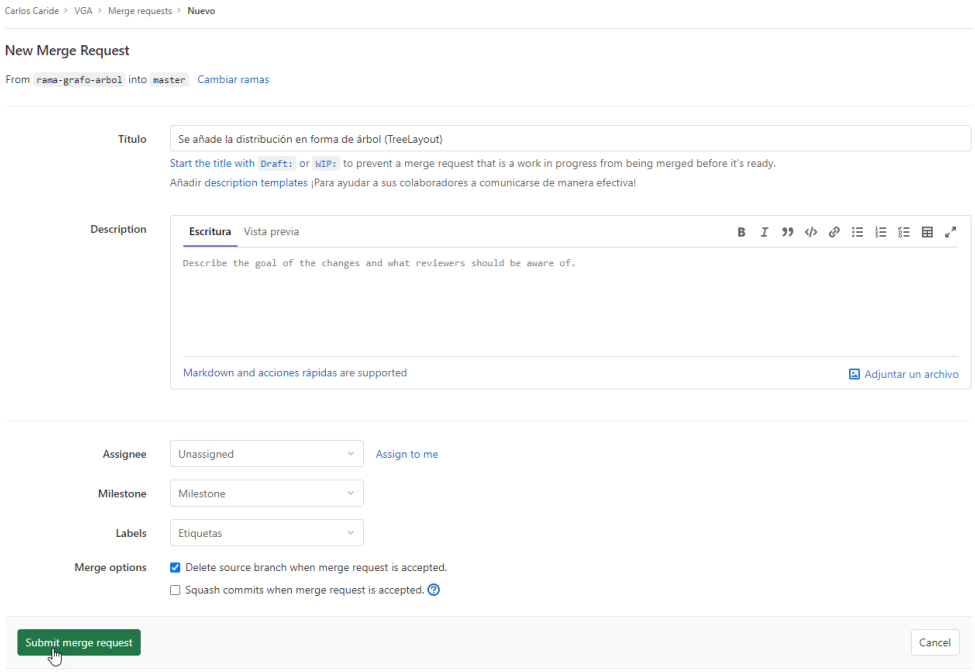


Figura 85: Creación de la solicitud de fusión

El desarrollador ya ha finalizado su parte. Ahora es turno de los administradores (mantainers) del repositorio, quienes son los autorizados a realizar fusiones. Para ello el administrador deberá de validar el trabajo y la corrección de la nueva funcionalidad. Si da el visto bueno, aprobará la solicitud y realizará la fusión de las ramas.

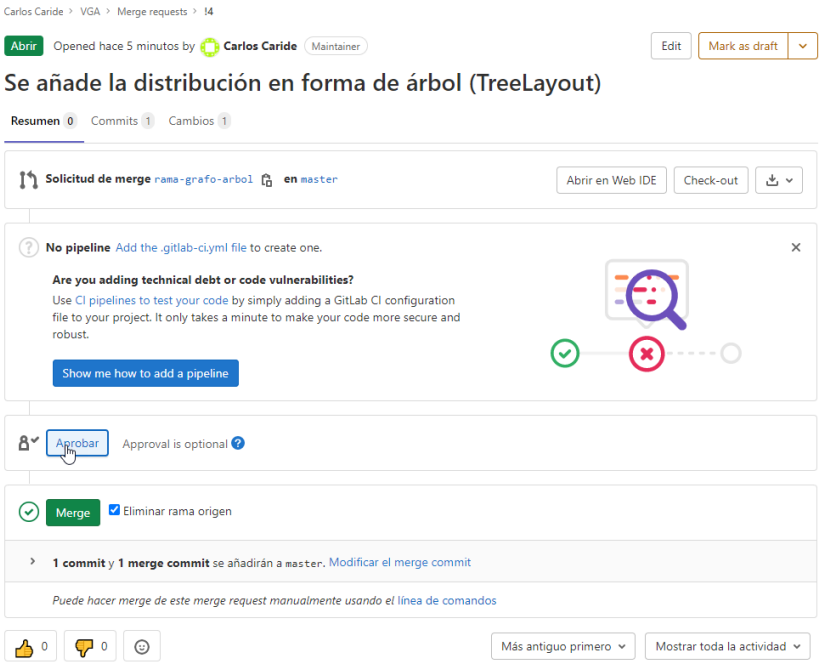


Figura 86: Aprobación de la fusión

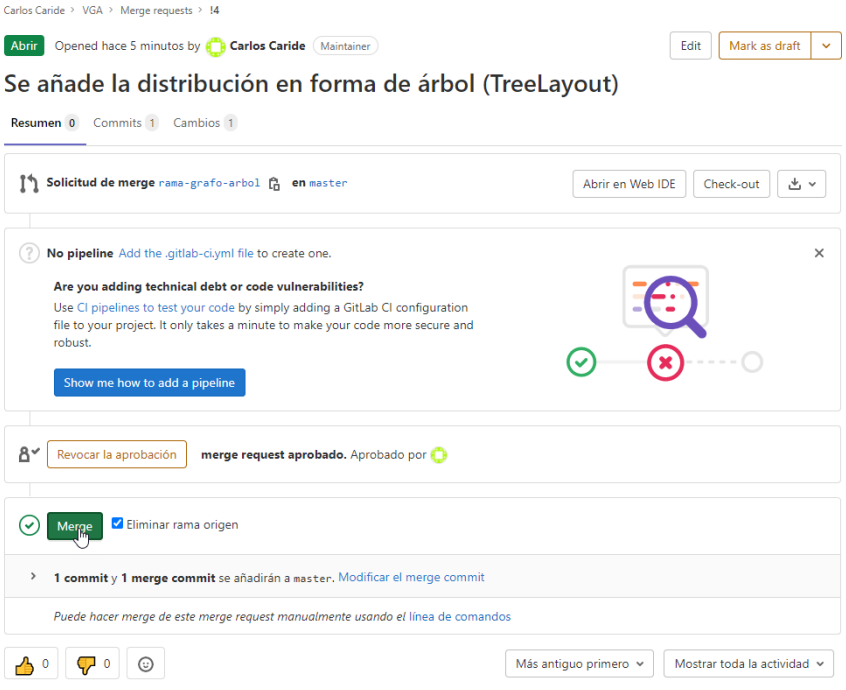


Figura 87: Ejecución de la fusión y eliminación de la rama

Finalmente, la rama *rama-grafo-arbol* habrá sido eliminada y master contendrá la nueva funcionalidad.

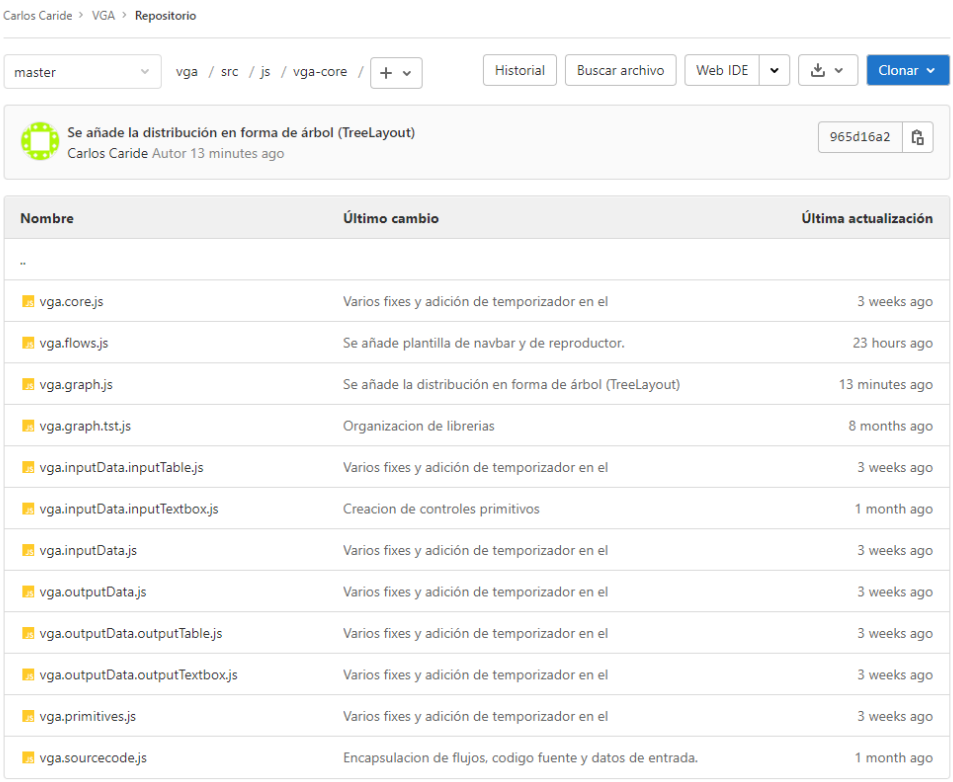


Figura 88: Situación final de master con la distribución en arbol añadida

10.3 Tutorial creación de algoritmo de ordenación Quicksort

En este anexo vamos a explicar paso a paso como se desarrolla un algoritmo en VGA.

El algoritmo escogido es el de ordenación, más concretamente Quicksort.

10.3.1 Creación de la petición y de la rama correspondientes

Para la mayor trazabilidad de los desarrollos, se recomienda que se realice una petición y el desarrollo en una nueva rama como se explica en las secciones crear solicitud y crear rama

10.3.2 Creación de los archivos necesarios

El primer paso es crearnos los nuevos archivos y carpetas que contendrán el nuevo algoritmo. En este caso, crearemos una carpeta con los archivos que se muestran en la imagen:

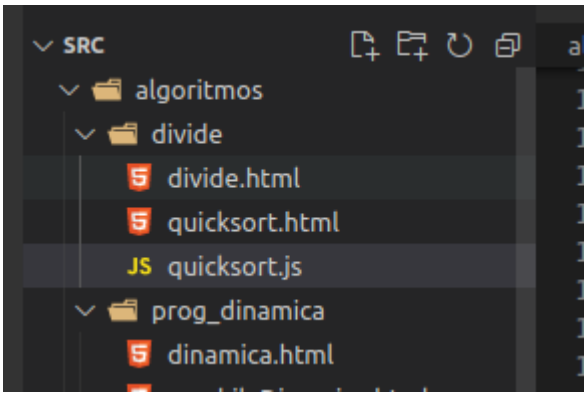


Figura 89: Estructuras de carpetas y archivos para el algoritmo Quicksort

Entonces pues, crearemos la carpeta divide, que contendrá los diferentes algoritmos de esta familia. Acto seguido, creamos tres archivos:

- **divide.html**: contendrá la información general de la familia de algoritmos y el enlace a los desarrollados.
- **quikcsort.html**: contendrá la información del algoritmo y será una copia del archivo *plantilla.html* modificando la literatura necesaria.

- **quicksort.js**: contendrá la lógica de funcionamiento del algoritmo. Al igual que el archivo html, será una copia de *plantilla.js*.

10.3.3 Modificación del menú principal

En este nuevo algoritmo se añade una nueva familia de algoritmos. Para dar más información se crea una nueva página que contiene información de esa familia. Para poder acceder a ella es necesario modificar el menú principal de VGA. Para ello, abrimos el archivo *menu.html*. Finalmente, insertaremos el enlace a la nueva página añadiendo el siguiente elemento:

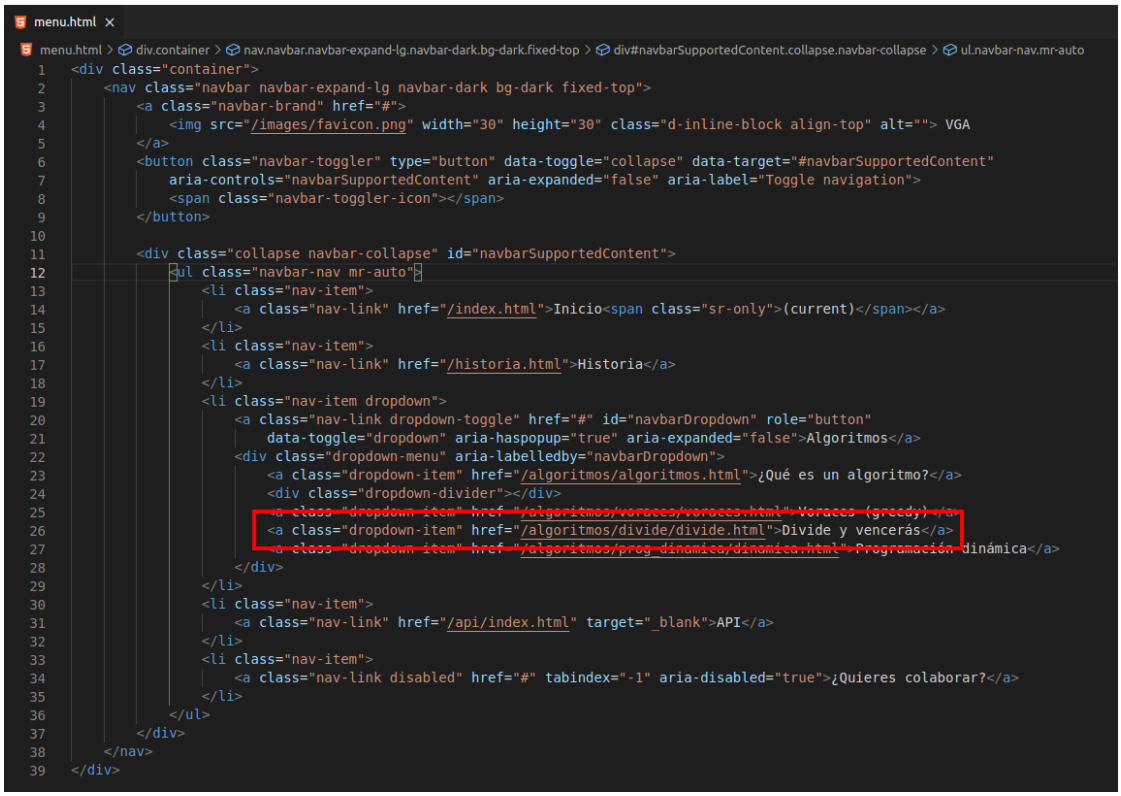


Figura 90: Añadir nuevo enlace a una página en el menú

10.3.4 Archivo divide.html

En este archivo se añade información relativa a la familia de algoritmos y los enlaces a los que se tiene implementados en VGA.

Figura 91: Vista de la página de divide y vencerás

En este link se puede ver el archivo con más detalle.

10.3.5 Archivo quicksort.html

El archivo contendrá la información teórica para la comprensión del algoritmo y será la página donde se ejecutará.

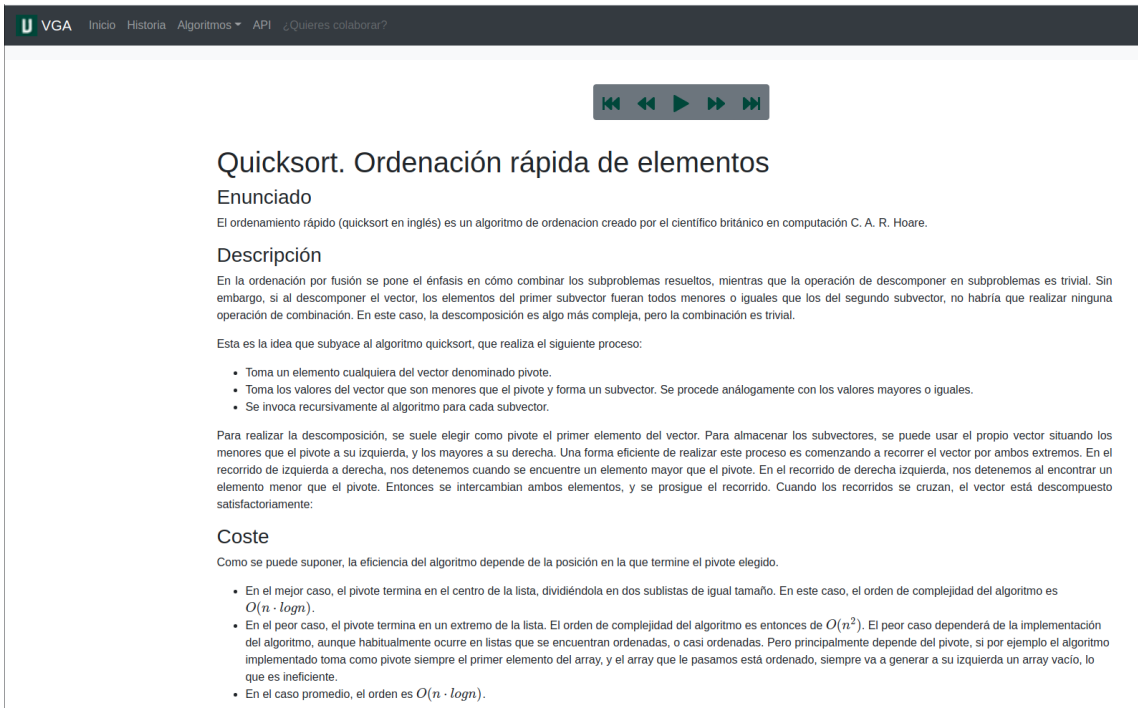


Figura 92: Página del algoritmo Quicksort

En este link se puede ver el archivo con más detalle.

10.3.6 Archivo quicksort.js

El archivo js es el encargado de ejecutar el algoritmo pero también de definir las claves del lenguaje utilizado y los campos de entrada y salida del algoritmo. La estructura básica se basa en las siguientes funciones:

```
function inicializar() {
  inicializarZonaVariables();

  codigoAlgoritmo();
  inicializarZonaResultados();
  ejecutarAlgoritmo();
}

//Imprime el algoritmo
function codigoAlgoritmo() {

}

//Ejecuta el algoritmo
function ejecutarAlgoritmo() {

}

//Inicia las variables
function inicializarZonaVariables() {
  InputData.onUpdate(actualizarDatos);
}
```

```
//Inicia para este caso la tabla, el objetos de volumen y el beneficio
function inicializarZonaResultados() {

}

//actualiza los datos
function actualizarDatos() {
  OutputData.clearResults();

  ejecutarAlgoritmo();
}
```

Estas funciones son las que procedemos a modificar para que el algoritmo se ejecute correctamente. En este link se puede ver el archivo con más detalle.

10.3.7 Envío de los cambios a GitLab

Una vez probado el algoritmo, deberemos de subirlo a GitLab como se comentó en el apartado de envío de cambios a GitLab y realizamos el merge request como se indicaba en solicitación de merge request.

Realizado este último paso, habremos fusionado la rama del algoritmo con master, realizando así la integración total del mismo en VGA.

11 API

En este capitulo se va a enumerar los distintos controladores, controles y utilidades que dispone VGA en el momento de la elaboración de esta memoria.

11.1 Controladores

Aquí se engloban distintas clases controladoras que se encargan del flujo del algoritmo y de los datos de entrada y de salida.

11.1.1 SourceCode

Clase que proporciona utilidades para la impresión de código fuente

11.1.1.1 setLanguageTokens

Establece las palabras reservadas del lenguaje

Parámetros

- `tokens` **Array<string>** Array de palabras reservadas del lenguaje

11.1.1.2 setLanguageSymbols

Símbolos que se reemplazaran por los códigos dados

Parámetros

- `simbolos` **Array<string>** Símbolos a modificar
- `codigos` **Array<string>** Códigos de reemplazo

11.1.1.3 printCodeLine

Imprime una línea de código en el bloque html reservado

Parámetros

- `codigo` **string** Línea de código fuente
- `indentado` **number** Número de indentado de la línea

11.1.1.4 printAuxLineCode

Imprime una línea de código en el bloque html indicado

Parámetros

- `codigo` **string** Línea de código fuente
- `indentado` **number** Número de indentado de la línea
- div **string** Elemento donde se imprimirá el código

11.1.2 MenuManager

Clase de utilidades para la automatización de menús.

11.1.2.1 createNavBar

Crea un menú de navegación entre las secciones de la página

11.1.3 ControlsContainer

Clase contenedora de los controles

11.1.3.1 addControl

Añade un nuevo control

Parámetros

- `control` **ControlPrimitive** Control a añadir

11.1.3.2 repaintControl

Repinta el control indicado

Parámetros

- `control` **ControlPrimitive** Control a repintar

11.1.3.3 get

Obtiene el nombre del control indicado

11.1.3.3.1 Parámetros

- `name` **string** Nombre del control

Valor de retorno: El control con nombre indicado

11.1.3.4 getControls

Obtiene todos los controles

Valor de retorno: Controles existentes

11.1.4 InputData

Extiende la clase ControlsContainer

Clase gestora de los controles de datos de entrada

11.1.4.1 addControl

Añade un control a la entrada de datos

Parámetros

- `control` **ControlPrimitive** Control a añadir

11.1.4.2 updateData

Actualiza los datos de los controles

11.1.4.3 onUpdate

Ejecuta una acción al evento de actualizar los datos

Parámetros

- `func` **function** Función a ejecutar al actualizar

11.1.5 OutputData

Extiende la clase ControlsContainer

Clase gestora de los controles de datos de salia

11.1.5.1 addControl

Añade un control a la salida de datos

Parámetros

- `control` **ControlPrimitive** Control a añadir

11.1.5.2 repaintControl

Repinta un control de la salia de datos

Parámetros

- `control` **ControlPrimitive** Control a repintar

11.1.5.3 clearResults

Limpia los resultados.

11.1.6 FlowManager

Clase gestora del flujo de ejecución del algoritmo

11.1.6.1 startFlow

Inicializa el gestor

11.1.6.2 finishFlow

Finaliza el gestor

11.1.6.3 startStep

Crea un nuevo paso en el gestor

11.1.6.4 finishStep

Finaliza el paso creado

11.1.6.5 reset

Vuelve al primer paso del algoritmo

11.1.6.6 back

Retrocede un paso en el algoritmo

11.1.6.7 next

Avanza un paso en el algoritmo

11.1.6.8 last

Va al último paso en el algoritmo

11.1.6.9 trogglePlay

Alterna el avance automático de pasos

11.1.6.10 includeAction

Añade una nueva acción que se ejecutará al entrar al paso

Parámetros

- `action` **action** Acción a añadir

11.1.6.11 includeBackAction

Añade una nueva acción que se ejecutará al entrar volver al paso anterior

Parámetros

- `action` **action** Acción a añadir

11.1.6.12 includeExitAction

Añade una nueva acción que se ejecutará al salir del paso hacia adelante

Parámetros

- `action` **action** Acción a añadir

11.1.6.13 newCellValue

Valor que se setará en la entrada del paso para la celda de la tabla indicada

Parámetros

- `table` **string** Nombre de la tabla
- `row` **number** Número de la fila

- `column` **number** Número de la columna
- `value` **object** Nuevo valor

11.1.6.14 oldCellValue

Valor que se seteará cuando se vuelva para el paso anterior del paso actual para la celda de la tabla indicada

Parámetros

- `table` **string** Nombre de la tabla
- `row` **number** Número de la fila
- `column` **number** Número de la columna
- `value` **object** Valor viejo

11.1.6.15 selectSourceLine

Selecciona la línea de código en la entrada del paso

Parámetros

- `line` **number** Número de línea

11.1.6.16 selectCell

Selecciona para ese paso la celda de la tabla indicada

Parámetros

- `table` **string** Nombre de la tabla
- `row` **number** Número de la fila
- `column` **number** Número de la columna

11.1.6.17 selectCellPermanent

Selecciona una celda permanentemente de la tabla indicada

Parámetros

- `table` **string** Nombre de la tabla
- `row` **number** Número de la fila
- `column` **number** Número de la columna

11.1.6.18 newResultValue

Valor que se seteará cuando se entre en el paso para la variable indicada

Parámetros

- `name` **string** Nombre de la variable
- `value` **object** Valor nuevo

11.1.6.19 oldResultValue

Valor que se seteará cuando se vuelva para el paso anterior para una variable dada

Parámetros

- `name` **string** Nombre de la variable
- `value` **object** Valor antiguo

11.1.6.20 setGraphStyle

Aplica el estilo al elemento del grafo indicado al entrar en el paso

Parámetros

- `graph` **string** Nombre del grafo
- `elementId` **string** Identificador de elemento o grupo
- `style` **object** Estilo a aplicar

11.1.6.21 unsetGraphStyle

Quita el estilo al elemento del grafo indicado al entrar en el paso

Parámetros

- `graph` **string** Nombre del grafo
- `elementId` **string** Identificador de elemento o grupo
- `style` **object** Esilo a quitar

11.1.6.22 newBarValue

Valor que se seteará en la entrada del paso para la barra del gráfico indicado

Parámetros

- `chart` **string** Nombre de la gráfico
- `id` **number** Número de barra
- `value` **object** Nuevo valor

11.1.6.23 oldBarValue

Valor que se seteará cuando se vuelva para el paso anterior del paso actual para la barra del gráfico indicado

Parámetros

- `chart` **string** Nombre de la gráfico
- `id` **number** Número de barra
- `value` **object** Valor viejo

11.1.6.24 selectBar

Selecciona para ese paso la barra del gráfico indicado

Parámetros

- `chart` **string** Nombre de la gráfico
- `id` **number** Número de barra

11.1.6.25 selectBarPermanent

Selecciona una barra permanentemente del gráfico indicado

Parámetros

- `chart` **string** Nombre de la gráfico
- `id` **number** Número de barra
- `color` (optional, default `"red"`) Color de la barra

11.1.6.26 unselectBar

Deselecciona para ese paso la barra del gráfico indicado

Parámetros

- `chart` **string** Nombre de la gráfico
- `id` **number** Número de barra

11.1.6.27 executeAction

Ejecuta la acción dada

Parámetros

- `actualAction` **action** Acción a ejecutar

11.1.6.28 selectLine

Acción de seleccionar una línea de código

Parámetros

- `line` **number** Línea a seleccionar

11.1.6.29 unselectLine

Acción de deseleccionar una línea de código

Parámetros

- `line` **number** Número de línea

11.2 Controles

Se presentan a continuación los distintos tipos de controles de los que se dispone en VGA.

11.2.1 ControlPrimitive

Clase primitiva de los controles a mostrar en la página.

11.2.1.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control

11.2.1.2 getHtmlCode

Obtiene el html del control

Valor de retorno: El código html del control

11.2.1.3 readControl

Lee los datos del control

11.2.1.4 getName

Retorna el nombre del control

Valor de retorno: El nombre del control

11.2.1.5 setLabel

Establece la etiqueta del control

Parámetros

- `label` **string** Etiqueta del control

11.2.1.6 getLabel

Obtiene la etiqueta del control

Valor de retorno: Etiqueta del control

11.2.1.7 setDescription

Establece la descripción del control

Parámetros

- `description` **string** Descripción del control

11.2.1.8 getDescription

Obtiene la descripción del control

Valor de retorno: Descripción del control

11.2.2 TextBoxPrimitive

Extiende la clase ControlPrimitive

Clase primitiva que representa un control de tipo caja de texto

11.2.2.1 Constructor

Parámetros

- `name` **string** Nombre del control

- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `data` **object** Valor del control

11.2.2.2 setData

Establece el valor del control

Parámetros

- `data` **object** Valor del control

11.2.2.3 getData

Retorna el valor del control

Valor de retorno: Valor del control

11.2.3 InputTextBox

Extiende la clase TextBoxPrimitive

Clase que representa un cuadro de texto de entrada

11.2.3.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `data` **object** Valor del control

11.2.3.2 getHtmlCode

Obtiene el html del control

Valor de retorno: El código html del control

11.2.3.3 readControl

Lee los datos del control

11.2.4 OutputTextBox

Extiende la clase TextBoxPrimitive

Clase que representa un cuadro de texto de salida

11.2.4.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `data` **object** Valor del control

11.2.4.2 getHtmlCode

Obtiene el html del control

Valor de retorno: El código html del control

11.2.4.3 clearData

Vacía el valor del control

11.2.5 TablePrimitive

Extiende la clase ControlPrimitive

Clase primitiva que representa una tabla

11.2.5.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `columnHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las columnas
- `columnHeaders` **Array<string>** Cabeceras personalizadas de las columnas
- `rowHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las filas
- `rowHeaders` **Array<string>** Cabeceras personalizadas de las filas
- `data`

11.2.5.2 setColumnHeaderType

Establece la cabecera de las columnas

Parámetros

- `type` **TABLEHEADERTYPE** Tipo de cabecera

11.2.5.3 getColumnHeaderType

Obtiene el tipo de cabecera de las columnas

Valor de retorno: El tipo de cabecera

11.2.5.4 setRowHeaderType

Establece la cabecera de las filas

Parámetros

- `type` **TABLEHEADERTYPE** Tipo de cabecera

11.2.5.5 getRowHeaderType

Obtiene el tipo de cabecera de las filas

Valor de retorno: El tipo de cabecera

11.2.5.6 setCustomColumnHeaders

Establece las cabeceras personalizadas de las columnas

Parámetros

- `headers`
- `columnHeaders` **Array<string>** Cabeceras personalizadas de las columnas

11.2.5.7 getCustomColumnHeaders

Obtiene las cabeceras personalizadas de las columnas

Valor de retorno: Cabeceras personalizadas de las columnas

11.2.5.8 setCustomRowHeaders

Establece las cabeceras personalizadas de las filas

Parámetros

- `headers`
- `columnHeaders` **Array<string>** Cabeceras personalizadas de las filas

11.2.5.9 getCustomRowHeaders

Obtiene las cabeceras personalizadas de las filas

Valor de retorno: Cabeceras personalizadas de las filas

11.2.5.10 setData

Establece los datos de la tabla

Parámetros

- `data`

11.2.5.11 getData

Obtiene los datos de la tabla

Valor de retorno: Datos de la tabla

11.2.5.12 setRowsTitle

Establece el nombre de la columna de las filas

Parámetros

- `rowsTitle` **string** El nombre de la columna

11.2.5.13 getRowsTitle

Obtiene el nombre de la columna de las filas

Valor de retorno: El nombre

11.2.6 InputTable

Extiende la clase TablePrimitive

Clase que representa una tabla de datos de entrada

11.2.6.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `columnHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las columnas
- `columnHeaders` **Array<string>** Cabeceras personalizadas de las columnas
- `rowHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las filas
- `rowHeaders` **Array<string>** Cabeceras personalizadas de las filas
- `data`

11.2.6.2 getHtmlCode

Obtiene el html de la tabla

Valor de retorno: El código html de la tabla

11.2.6.3 readControl

Lee los datos de la tabla

11.2.6.4 addRow

Añade una nueva fila

11.2.6.5 deleteRow

Elimina la fila indicada

Parámetros

- `row` **number** Fila a eliminar

11.2.6.6 setStatic

Establece si se puede añadir/eliminar filas a la tabla

Parámetros

- `staticTable` **boolean** True si no se permite añadir/eliminar filas

11.2.7 OutputTable

Extiende la clase TablePrimitive

Clase que representa una tabla de datos de salida

11.2.7.1 Constructor

Parámetros

- `name` **string** Nombre del control
- `label` **string** Etiqueta del control
- `description` **string** Descripción del control
- `columnHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las columnas
- `columnHeaders` **Array<string>** Cabeceras personalizadas de las columnas
- `rowHeaderType` **TABLEHEADERTYPE** Tipo de cabecera de las filas
- `rowHeaders` **Array<string>** Cabeceras personalizadas de las filas
- `data`

11.2.7.2 getHtmlCode

Obtiene el html de la tabla

Valor de retorno: El código html de la tabla

11.2.7.3 setValue

Establece el valor indicado en la tabla

Parámetros

- `row` **number** Número de fila
- `column` **number** Número de columna
- `value` **object** Valor

11.2.7.4 selectCell

Selecciona la celda de la tabla

Parámetros

- `row` **number** Número de fila
- `column` **number** Número de columna

11.2.7.5 unselectCell

Deselecciona la celda de la tabla

Parámetros

- `row` **number** Número de fila
- `column` **number** Número de columna

11.2.7.6 clearData

Vacía el contenido de la tabla

11.2.8 Graph

Extiende la clase ControlPrimitive

Clase que representa un grafo

11.2.8.1 Constructor

11.2.8.1.1 Parámetros

- `name` **string** Nombre del grafo
- `label` **string** Etiqueta que se mostrará
- `description` **string** Descripción que se mostrará

11.2.8.2 getHtmlCode

Retorna el html asociado a este control

Valor de retorno: Código html del control

11.2.8.3 initGraph

Inicia el grafo con los estilos asociados

11.2.8.3.1 Parámetros

- `styles` **object** Estilos del grafo

11.2.8.4 addNode

Añade un nuevo nodo al grafo

11.2.8.4.1 Parámetros

- `idNode` **string** Identificador del nodo
- `label` **srting** Etiqueta del nodo

11.2.8.5 addEdge

Añade una nueva arista al grafo

11.2.8.5.1 Parámetros

- `idEdge` **string** Identificador de la arista
- `idOrigin` **string** Identificador del nodo origen
- `idDestiny` **string** Identificador del nodo destino
- `label` **string** Etiqueta de la arista

11.2.8.6 getElement

Obtiene el elemento del grafo

11.2.8.6.1 Parámetros

- `idElement` **string** Identificador del elemento

Valor de retorno: El elemento del grafo

11.2.8.7 checkIfExistElement

Comprueba que existe un elemento dado en el grafo

11.2.8.7.1 Parámetros

- `idElement` **string** Identificador del elemento

Valor de retorno: true si el elemento existe, false en otro caso

11.2.8.8 setStyleToElement

Aplica un estilo a un elemento dado

11.2.8.8.1 Parámetros

- `idElement` **string** Identificador del elemento
- `style` **string** Nombre del estilo a aplicar

11.2.8.9 unsetStyleToElement

Quita un estilo a un elemento dado

11.2.8.9.1 Parámetros

- `idElement` **string** Identificador del elemento
- `style` **string** Nombre del estilo a quitar

11.2.8.10 setLayout

Aplica una nueva distribución al grafo

11.2.8.10.1 Parámetros

- `layout` **Layout** Layout a aplicar

11.2.8.11 getLayout

Devuelve la distribución actual

Valor de retorno: El layout actual

11.2.8.12 refreshLayout

Refresca la distribución actual

11.2.8.13 clearData

Vacía el grafo

11.2.9 BarChart

Extiende la clase ControlPrimitive

Clase que representa una gráfica.

11.2.9.1 Constructor

11.2.9.1.1 Parámetros

- `name` **string** Nombre de la gráfica.
- `label` **string** Etiqueta que se mostrará.
- `description` **string** Descripción que se mostrará.

11.2.9.2 getHtmlCode

Retorna el html asociado a este control

Valor de retorno: Código html del control.

11.2.9.3 initChart

Inicia la gráfica.

11.2.9.3.1 Parámetros

- `dataset` **Array** Array de datos.

11.2.9.4 setData

Establece los datos del gráfico

11.2.9.4.1 Parámetros

- `data` **Array<object>** Datos del gráfico.

11.2.9.5 getData

Obtiene los datos del gráfico.

Valor de retorno: Datos del gráfico.

11.2.9.6 selectBar

Selecciona una barra.

11.2.9.6.1 Parámetros

- `id` **number** Identificador de la barra
- `color` **string** Nombre del color del borde. Rojo por defecto (opcional, por defecto `"red"`)

11.2.9.7 setBarStyle

Setea el estilo de una barra

11.2.9.7.1 Parámetros

- `id` **number** Identificador de la barra
- `style` **string** Estilo que se quiere poner a la barra

11.2.9.8 removeBarStyle

Elimina el estilo de una barra

11.2.9.8.1 Parámetros

- `id` **number** Identificador de la barra

11.2.9.9 unselectBar

Deselecciona una barra

11.2.9.9.1 Parámetros

- `id` **number** Identificador de la barra

11.2.9.10 getBarStyle

Obtiene el estilo de una barra

11.2.9.10.1 Parámetros

- `id` **number** Identificador de la barra

Valor de retorno: El estilo de la barra en caso de existir

11.2.9.11 setBarValue

Setea el valor de una barra

11.2.9.12 Parámetros

- `id` **number** Identificador de la barra
- `value` **number** Valor de la barra

11.2.9.13 clearData

Vacía la gráfica

11.3 Utilidades

Se enumeran las distintas clases, y enumerados que son usados en la herramienta.

11.3.1 Utilities

Clase que contiene distintas utilidades

11.3.1.1 refreshMath

Actualiza las fórmulas de la página

11.3.1.2 printArray

Devuelve un array como un string encerrado en corchetes { }

Parámetros

- `array` **Array<object>**

Valor de retorno: El array como string

11.3.1.3 getRandomInt

Devuelve un entero conprendido entre dos números

Parámetros

- `min` **number** Menor entero
- `max` **number** Mayor entero

Valor de retorno: El entero seleccionado aleatoriamente

11.3.1.4 getRandomInt

Devuelve un número comprendido entre otros dos

Parámetros

- `min` **number** Menor número
- `max` **number** Mayor número

Valor de retorno: El número seleccionado aleatoriamente

11.3.1.5 hasAttr

Comprueba si existe el atributo indicado para el elemento id

Parámetros

- `id` **string** Identificador del elemento
- `attr` **string** Atributo a comprobar

Valor de retorno: Verdadero en caso de que contenga el atributo indicado

11.3.2 TABLEHEADERTYPE

Enumerado con los tipos de encabezado de tablas

Type: number

11.3.2.1 NONE

Ningun encabezado

11.3.2.2 NUMBER

Encabezado de tipo numérico

11.3.2.3 NUMBER0

Encabezado de tipo numérico base 0

11.3.2.4 CUSTOM

Encabezado personalizado

11.3.3 Layout

Clase abstracta de una distribución de nodos en el grafo

11.3.3.1 Constructor

Parámetros

- `layoutType` **string** Nombre de la distribución

11.3.3.2 getLayoutType

Obtiene el nombre de la distribución

Valor de retorno: El nombre del layout

11.3.3.3 setLayoutType

Establece el nombre de la distribución

Parámetros

- `layoutType` **string** Establece el nombre del layout

11.3.4 RandomLayout

Extiende la clase Layout

Clase que representa una distribución aleatoria de los nodos del grafo

11.3.4.1 getLayout

Crea el layout del grafo

Parámetros

- `graph` **Graph** Grafo al que se le creará el layout

11.3.5 GridLayout

Extiende la clase Layout

Clase que representa la distribución en forma de tabla en un grafo

11.3.5.1 Constructor

Parámetros

- `rows` **number** Número de filas
- `columns` **number** Número de columnas

11.3.5.2 getRows

Retorna el número de filas de la distribución

Valor de retorno: Número de filas

11.3.5.3 setRows

Establece el número de filas de la distribución

Parámetros

- `rows` **number** Número de filas

11.3.5.4 getColumns

Retorna el número de columnas de la distribución

Valor de retorno: Número de columnas

11.3.5.5 setColumns

Establece el número de columnas de la distribución

Parámetros

- `columns`
- `rows` **number** Número de columnas

11.3.5.6 getLayout

Crea el layout del grafo

Parámetros

- `graph` **Graph** Grafo al que se le creará el layout

11.3.6 TreeLayout

Extiende la clase Layout

Clase que representa la distribución en forma de árbol en un grafo

11.3.6.1 getLayout

Crea el layout del grafo

Parámetros

- `graph` **Graph** Grafo al que se le creará el layout