ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2023/2024

Trabajo Fin de Grado

REDISEÑO Y AMPLIACIÓN DEL SISTEMA DE EXPERIMENTACIÓN ALGOREX

Autor: Juan Rodríguez Alonso

Director: Jesús Ángel Velázquez Iturbide

Agradecimientos

Me gustaría darle las gracias a mis padres y a mi hermano, por haber sido mi apoyo y soporte durante toda mi carrera, especialmente en los momentos más difíciles.

También me gustaría darle las gracias a todos los compañeros que he tenido a lo largo de la carrera, pues en su gran mayoría se han convertido en grandes amigos que me quedo para toda la vida.

Por último, y no menos importante, me gustaría darle las gracias a los profesores que he tenido a lo largo de la carrera, pues han incentivado las ganas y la pasión que tengo por el mundo de la informática, y en especial, el de la programación.

Resumen

El sistema AlgorEx es una aplicación educativa desarrollada en Java que permite experimentar y comparar la optimalidad y la eficiencia en tiempo de algoritmos que resuelven un mismo problema, mostrando los resultados mediante tablas y gráficos.

Este trabajo de fin de grado consiste en aplicar diferentes mejoras y nuevas funcionalidades al sistema AlgorEx, con el fin de hacer más sencillo al usuario la experimentación con juegos de datos.

Estos cambios consisten en:

- Funcionamiento de AlgorEx en diferentes JDK (de la versión 6 en adelante).
- Mejorar la reordenación de las tablas de datos y ejecuciones de forma coordinada.
- Mejorar varios aspectos ya existentes de la generación aleatoria de datos.
- Integración de nuevos generadores de datos:
    - Generador de tareas de duración no unitaria.
    - Generador de permutaciones.
    - Generador de grafos por probabilidad de arco.
    - Generador de grafos por número de arcos.
    - Generador de grafos por número de etapas.

Los cambios realizados se han llevado a cabo mediante una metodología centrada en el usuario, evaluando la funcionalidad y usabilidad del sistema.

Palabras clave: AlgorEx, experimentación, ordenación de tablas, juego de datos, generador de datos, tareas, permutaciones, grafos.

Índice

Índice de Figuras

1. Introducción

1.1. Motivación

En el mundo de la programación, la optimalidad y la eficiencia de los algoritmos han sido siempre temas elementales, los cuales han sufrido una constante evolución. Esto es así debido a que hacen referencia a la capacidad que tiene un algoritmo para poder resolver un problema de forma óptima y en una cantidad de tiempo que sea razonable.

Con el paso de los años y la correspondiente evolución de la tecnología, la capacidad de procesamiento de los computadores ha ido aumentando y, por ende, los enfoques para lograr la optimalidad y la mayor eficiencia posible se han vuelto más sofisticados.

De igual manera, las herramientas que permiten medir estos parámetros han ido evolucionando y han ido ofreciendo una mayor cantidad de información y una mejor experiencia de usuario al desarrollador, con el fin de que sea más sencillo hacer más eficientes aquellos algoritmos que permiten abordar y resolver problemas complejos.

Es en este punto en el que entran sistemas como AlgorEx[10]. Dado que el sistema AlgorEx es utilizado tanto por profesores como alumnos en asignaturas del grado como Algoritmos Avanzados, como fue en mi caso, me causó bastante interés el hecho de ampliar y mejorar las funcionalidades de este sistema.

El sistema AlgorEx es una aplicación educativa desarrollada en Java, mediante la cual es posible experimentar y comparar la optimalidad y la eficiencia en tiempo de los diferentes algoritmos que resuelven un mismo problema, mostrando los resultados a través de tablas y gráficos.

En este sistema, para poder realizar la experimentación y comparar la optimalidad y eficiencia en tiempo de los algoritmos, era posible generar *n* juegos de datos (ejemplos de prueba) de manera aleatoria, los cuales se probaban en sucesivas ejecuciones. Sin embargo, la generación de dichos juegos de datos estaba limitada a los tipos básicos de Java, incluyendo *arrays* (tanto de una como de dos dimensiones).

Es por ello por lo que resultaba interesante el hecho de ampliar el número de generadores del sistema, pues disponía únicamente de un generador y, en determinados casos, se requieren más parámetros a modificar para la generación de los juegos de datos.

Así pues, se proponía desarrollar nuevos generadores de datos: uno para la generación de permutaciones (para los *arrays*), otro para la generación de tareas (para problemas como el de la Selección de Actividades) y otros tantos para la generación de grafos (en base a la probabilidad de arco, en base al número de arcos y en base al número de etapas).

Además de estos nuevos generadores, también eran necesarios algunos retoques en el generador de datos original y en el funcionamiento de las tablas de datos y de ejecuciones.

1.2. Objetivos

El principal objetivo de este trabajo de fin de grado ha sido implementar mejoras en el sistema AlgorEx. Las mejoras propuestas son las siguientes:

- Comprobar que AlgorEx funciona con diferentes versiones de JDK (Java Development Kit), desde la versión 6 en adelante.

- Mejorar la reordenación de las tablas de datos y ejecuciones de forma coordinada.

- Mejorar varios aspectos ya existentes de la generación aleatoria de datos.

- Inclusión de nuevos generadores de datos:

    - **Generador de tareas de duración no unitaria:** permite generar pares de números para los instantes de inicio y fin de tareas en una función, almacenándolos en dos vectores de enteros.

    - **Generador de permutaciones:** permite generar una permutación en un vector de enteros, que contiene elementos del 0 al n-1.

    - **Generadores de grafos:** permiten generar una matriz de enteros como una matriz de adyacencia de un grafo en una función.

      Para este caso, se deben crear tres nuevos generadores de datos:

        - **Generador de grafos en base a la probabilidad de arco:** genera un grafo con un número de nodos, probabilidad de arco, dirección y valuación definidos.

        - **Generador de grafos en base al número de arcos:** similar al anterior, pero genera el grafo en función del número de arcos en lugar de la probabilidad de arco.

        - **Generador de grafos multietapa:** genera un grafo con un número de nodos y número de etapas definidos (siendo éste valuado).

1.3. Estructura de la Memoria

Una vez realizada la introducción al proyecto, en esta parte se indicarán la estructura y los pasos a seguir para completar el contenido de esta memoria.

En primer lugar, se mostrará un apartado equivalente a lo que sería una *Sesión de Usuario*, en la que se probarán diferentes tareas y se adjuntarán capturas de pantalla que muestren el funcionamiento del sistema AlgorEx en distintas situaciones.

A continuación, se expondrá un apartado de *Metodología*, en el cual se mostrará una especificación completa de los objetivos y mejoras desarrolladas en este trabajo, además del proceso de desarrollo seguido para cumplir dichos objetivos y, por último, las herramientas utilizadas para ello.

Tras ello, se adjuntará un apartado de *Diseño*, en el que se mostrarán los diferentes cambios y mejoras desarrollados en la interfaz de usuario, además de la arquitectura de la aplicación (de las clases modificadas únicamente).

El siguiente paso será incluir un apartado de *Implementación*, en el cual se expondrá la explicación de los algoritmos utilizados para desarrollar las mejoras requeridas en los objetivos, incluyendo fragmentos de código representativos.

A continuación, se incluirá un apartado de *Evaluación*, en el que se explicarán las pruebas realizadas y el porqué de estas. Además, se mostrará un listado con las diferentes versiones desarrolladas del sistema y los cambios y mejoras que contenía cada una de ellas.

Por último, se expondrá un apartado de *Conclusiones y Trabajos Futuros*, en el que se hará una valoración personal de lo que ha supuesto el proyecto para mí, además de una estimación del tiempo empleado en el desarrollo. Junto a ello, se indican las posibles mejoras que podría tener este proyecto en un futuro.

Aparte de estos apartados, se dispone de tres apartados más que sirven para complementar lo explicado en la memoria: el apartado de *Bibliografía* y los apartados de *Anexos*: el *Anexo I*, en el que se incluye el código mencionado en el apartado de *Implementación* ; y el *Anexo II*, que se trata de un manual de configuración enfocado al desarrollador de cara a ampliaciones futuras del proyecto.

2. Sesión de Usuario

En este apartado de la memoria, se mostrarán diferentes casos de uso del sistema AlgorEx, enfocados principalmente a las mejoras desarrolladas sobre el proyecto.

En este caso, se verá el funcionamiento del generador de permutaciones y del generador de grafos en base a la probabilidad de arco.

Para el generador de permutaciones, probaremos el funcionamiento de la clase RedesCruzadas, pues tiene diferentes métodos para resolver el problema del subconjunto no cruzado de redes[13], los cuales contienen un *array* del que se pueden obtener permutaciones.

En primer lugar, tras abrir AlgorEx y cargar el fichero Java a probar (RedesCruzadas.java), seleccionamos los métodos a probar mediante el selector de problema, escogiendo las opciones de Optimalidad como criterio de experimentación, Maximizar como objetivo del problema y medidas relativas (véase Figura 1).

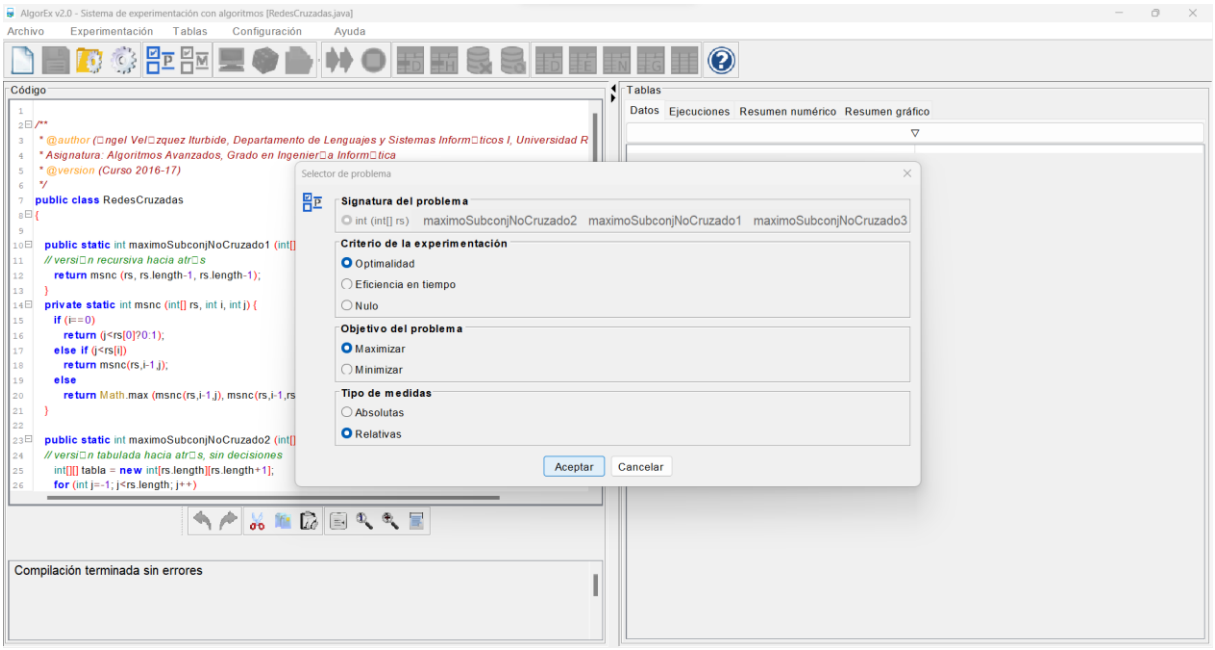


Figura 1: Diálogo del selector de problema (Caso de Uso - Generador de Permutaciones)

Tras ello, pulsamos el botón de Generar datos aleatorios (representado en la barra de iconos por un dado) para introducir el número de juegos de datos a generar. En este caso, elegiremos 20 juegos de datos (véase Figura 2).

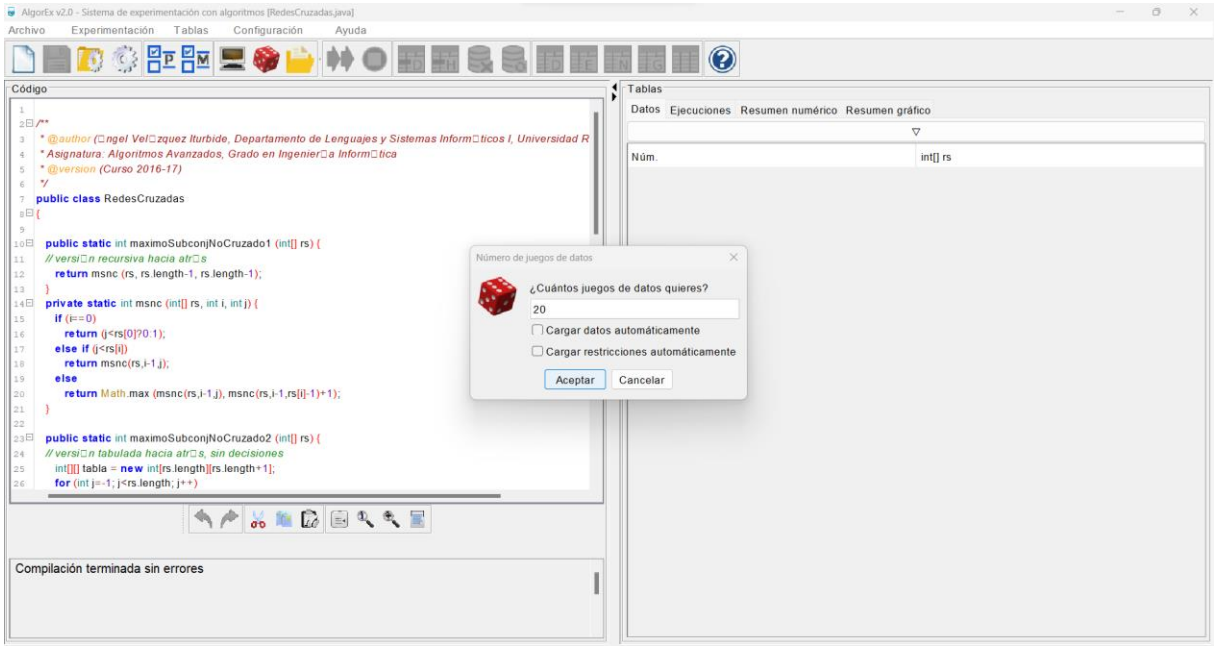


Figura 2: Diálogo del selector de número de juegos de datos (Caso de Uso - Generador de Permutaciones)

En el diálogo de Intervalos de aleatoriedad (del generador de datos principal), junto a los botones de Aceptar, Cancelar y Guardar, tenemos la opción de Perms. (para acceder al diálogo del generador de permutaciones). En este caso, elegiremos la opción de Perms. (véase Figura 3).

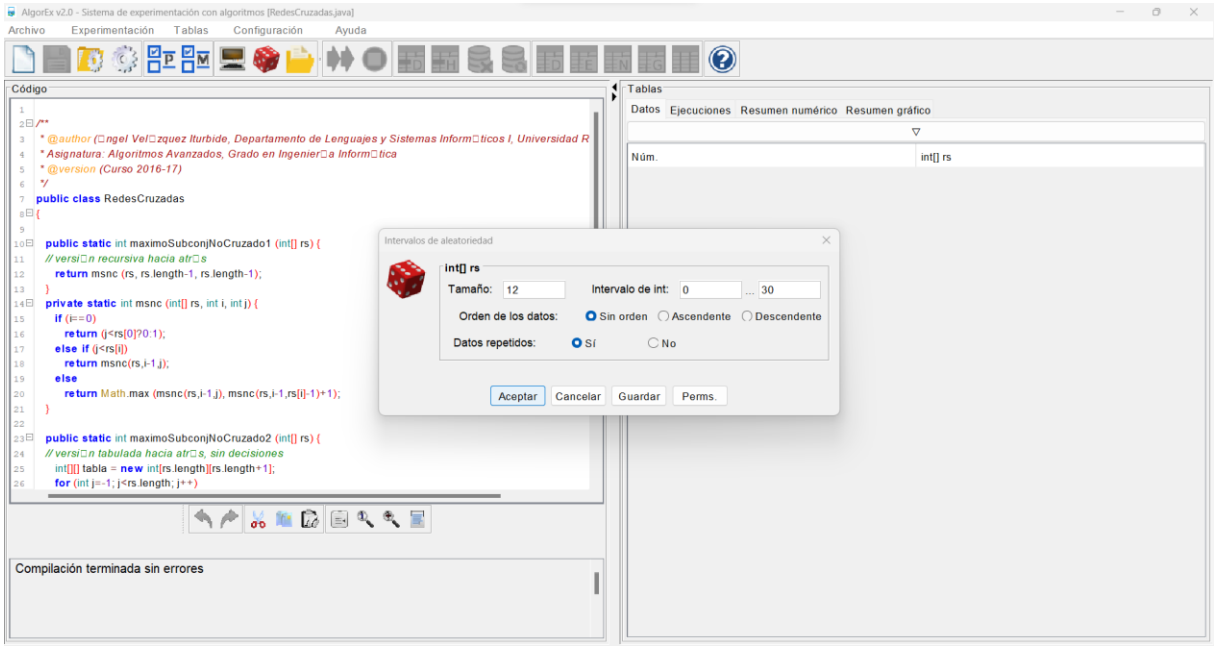


Figura 3: Diálogo del generador de datos principal (Caso de Uso - Generador de Permutaciones)

Dentro de las opciones que ofrece el diálogo del generador de permutaciones, podemos elegir el tamaño del *array* (teniendo en cuenta que, mediante este generador, se generan

permutaciones en base al tamaño del *array*, es decir, se generan *arrays* con elementos desde 0 hasta el tamaño menos uno).

En este caso, elegiremos que el *array* sea de tamaño 5 (véase Figura 4).

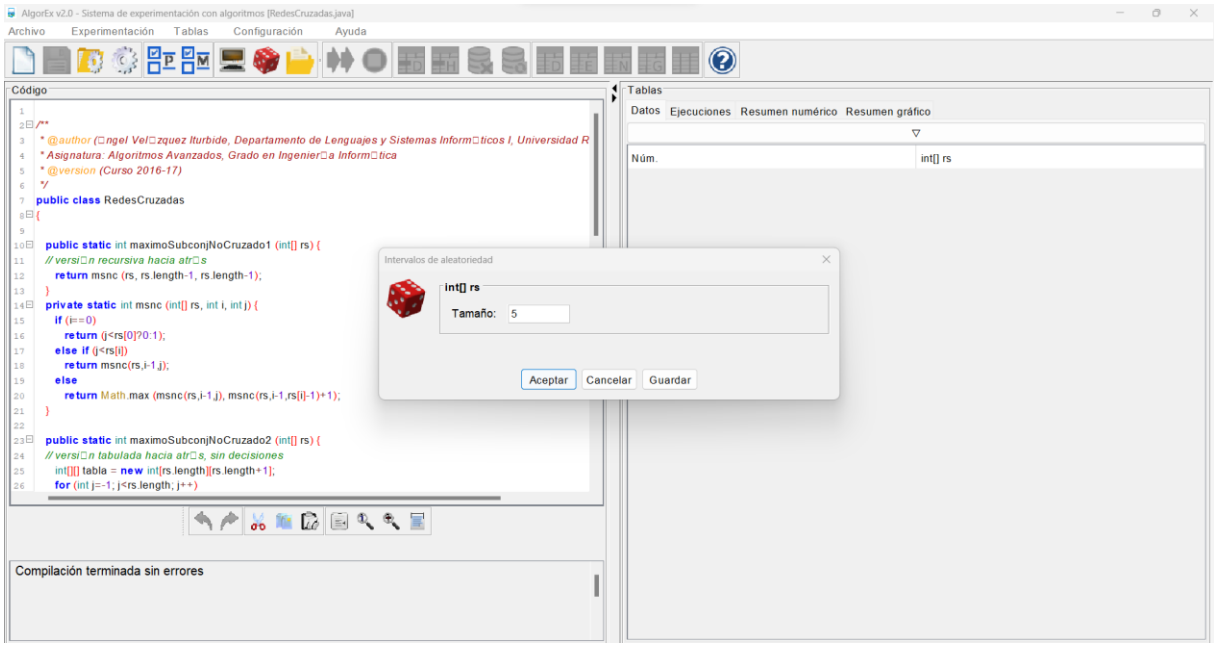


Figura 4: Diálogo del generador de permutaciones (Caso de Uso - Generador de Permutaciones)

Una vez aceptadas las restricciones, se generan 20 juegos de datos, generando para ellos permutaciones de 0 a 4 (véase Figura 5)

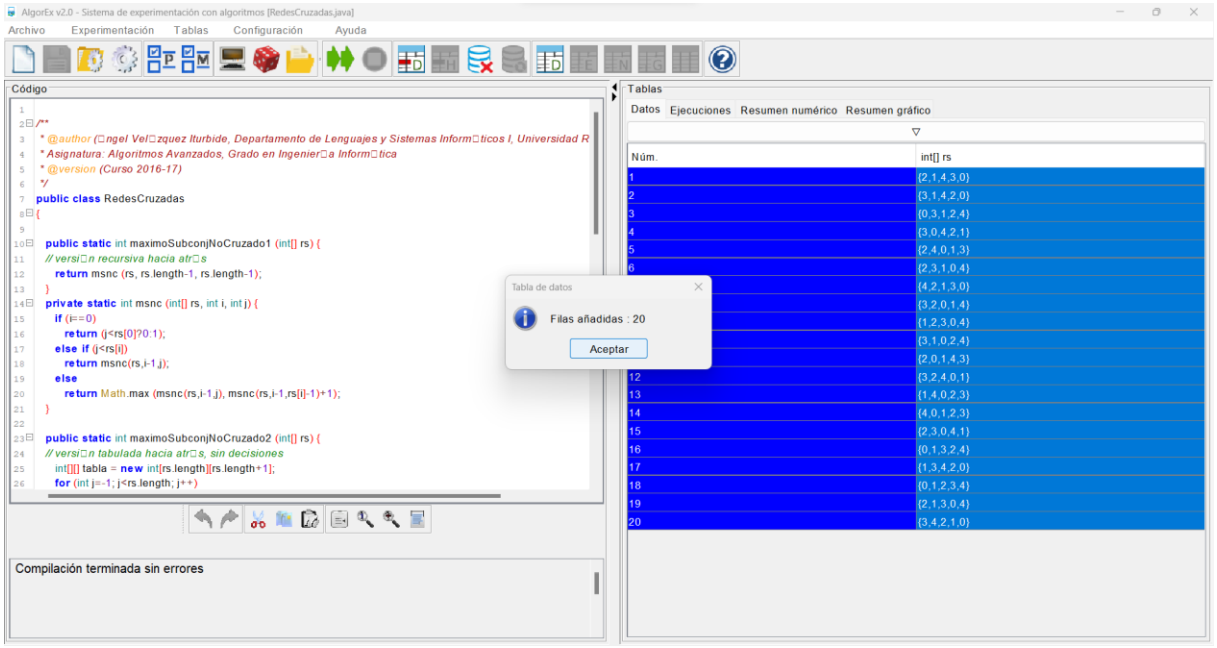


Figura 5: Resultado de la generación de juegos de datos (Caso de Uso - Generador de Permutaciones)

Tras haber generado los juegos de datos, se ejecutan, obteniendo los siguientes resultados (véase Figura 6).

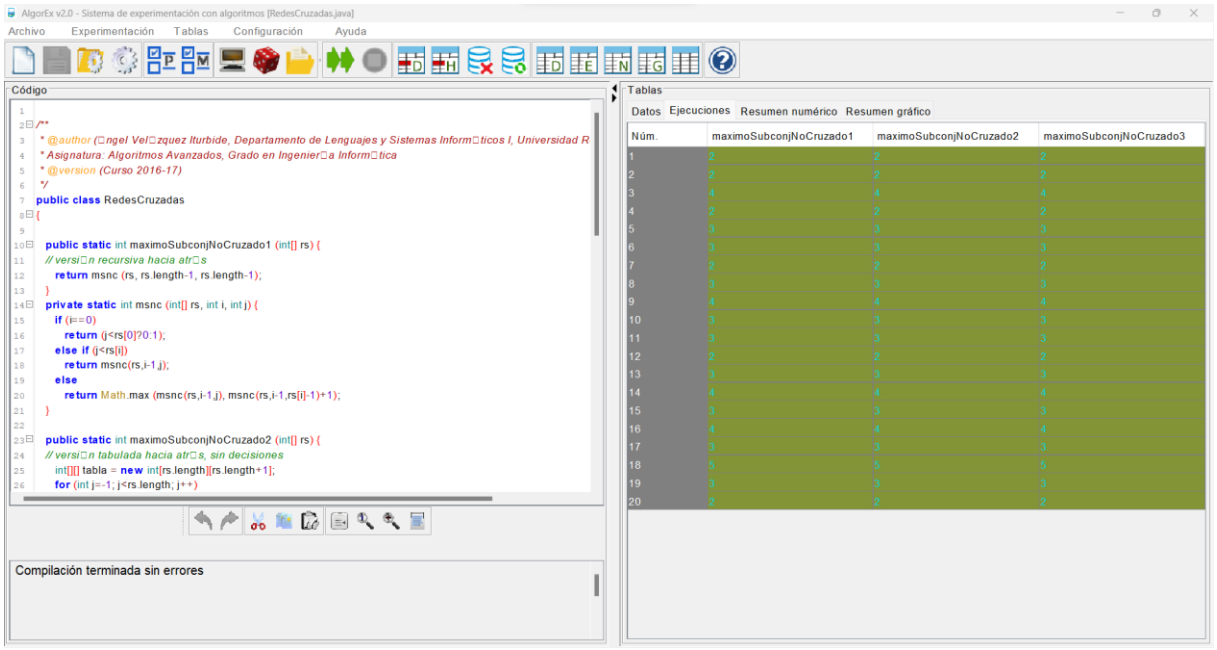


Figura 6: Tabla de ejecuciones (Caso de Uso - Generador de Permutaciones)

Una de las nuevas funcionalidades implementadas ha sido poder ordenar la tabla de ejecuciones por cualquiera de las columnas.

La ordenación por columnas permite ordenar las ejecuciones en función de los resultados obtenidos o en función de los tiempos de ejecución empleados para realizar dichas ejecuciones, todo ello dependiendo del criterio de experimentación activo (optimalidad o eficiencia en tiempo, respectivamente).

Además, en el caso de las tablas de datos y de ejecuciones, la reordenación de cada una de ellas por cualquiera de sus columnas se reproduce en la otra tabla y viceversa.

En este caso, se ordenan las ejecuciones en función de los beneficios calculados por el algoritmo de la función maximoSubconjNoCruzado2, viendo cómo se reordena también en la tabla de datos (véase Figuras 7 y 8).

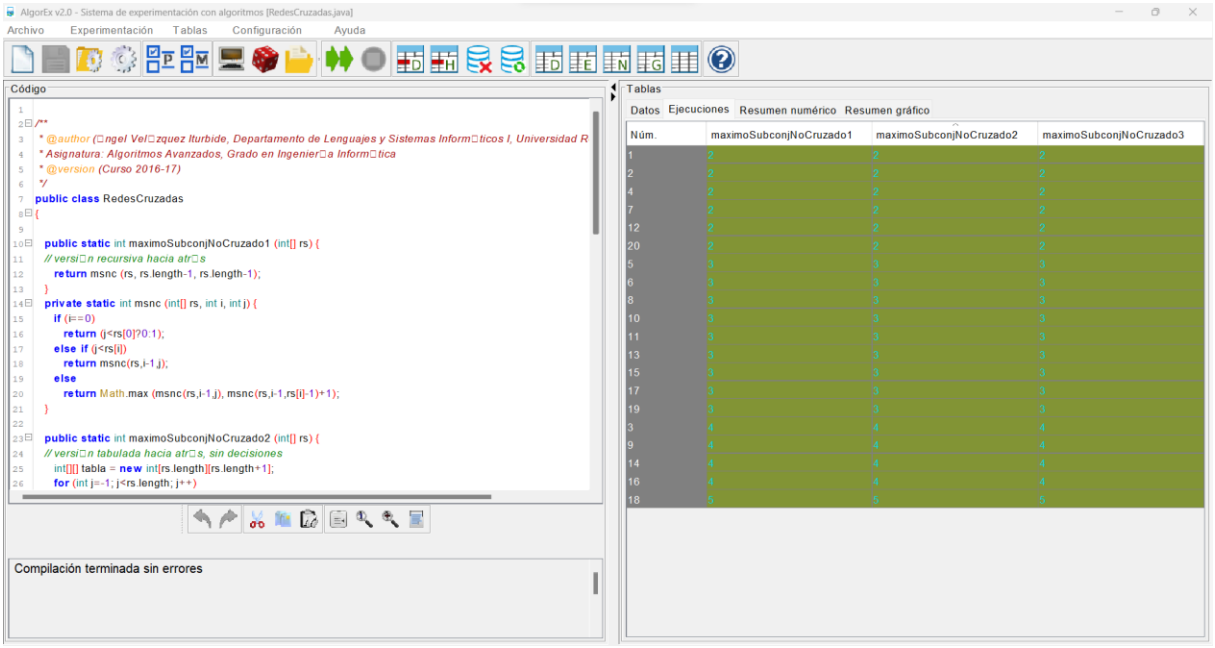


Figura 7: Tabla de ejecuciones tras reordenación (Caso de Uso - Generador de Permutaciones

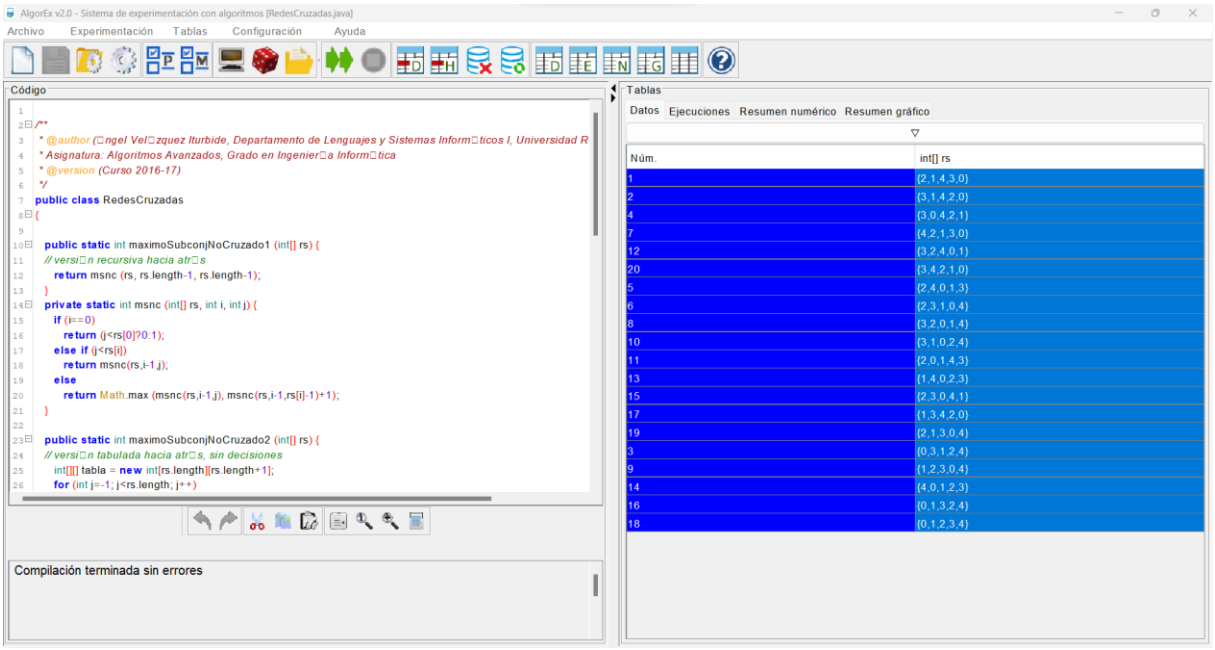


Figura 8: Tabla de datos tras la reordenación de la tabla de ejecuciones (Caso de Uso - Generador de Permutaciones)

Tras haber probado el generador de permutaciones, vamos a cargar el fichero CaminosMinimosDesdeUnNodo.java, pues dispone de métodos que nos permiten probar el generador de grafos en base a la probabilidad de arco.

En primer lugar, tras cargar el fichero Java, seleccionamos las funciones a probar mediante el selector de problema, seleccionando la opción de Eficiencia en tiempo como criterio de experimentación.

Tras ello, al igual que antes, pulsamos el botón de Generar datos aleatorios (representado en la barra de iconos por un dado) para introducir el número de juegos de datos a generar, volviendo a elegir 20.

En el diálogo de Intervalos de aleatoriedad (del generador de datos principal), tenemos las opciones de Grafo Probs. (corresponde al generador de grafos por probabilidad de arco), la de Grafo Arcos (corresponde al generador de grafos por arcos) y la de Grafo Etapas (corresponde al generador de grafos multietapa). Elegimos la de Grafo Probs. (véase Figura 9).

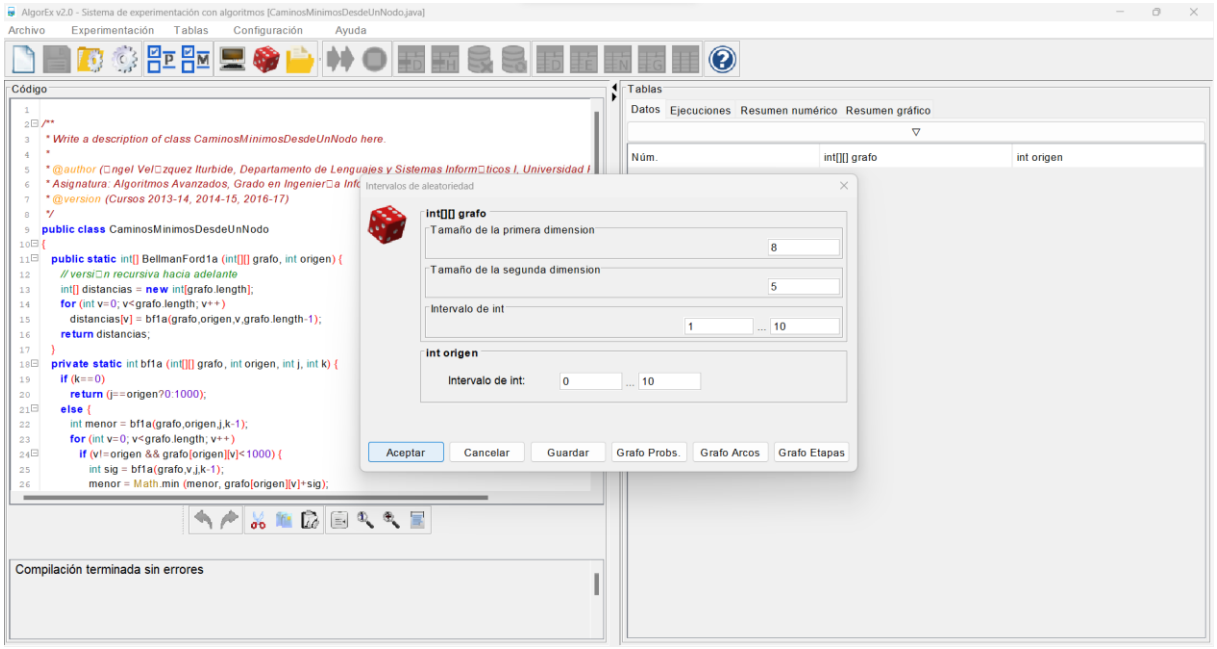


Figura 9: Diálogo del generador de datos principal (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

En el diálogo de este generador, se puede elegir el número de nodos, la probabilidad de arco, la dirección del grafo y si el grafo es valuado (o no), además de elegir el intervalo del entero. Para este caso, elegiremos que el número de nodos sea 4, la probabilidad de arco sea 0.75, el grafo sea no dirigido, no valuado y no reflexivo, y el entero esté entre 0 y 3 (véase Figura 10).

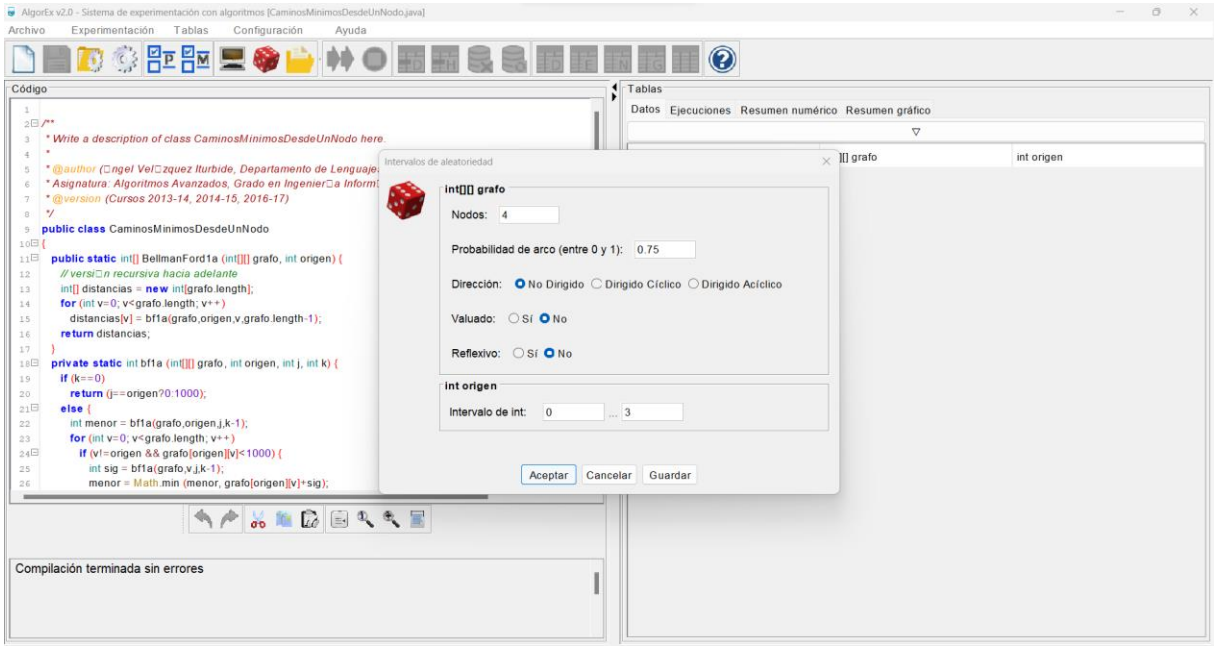


Figura 10: Diálogo del generador de grafos por probabilidad de arco (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

Al tratarse de un nuevo generador de datos, probamos la opción de guardar restricciones, incorporada a los nuevos generadores de datos (véase Figura 11).

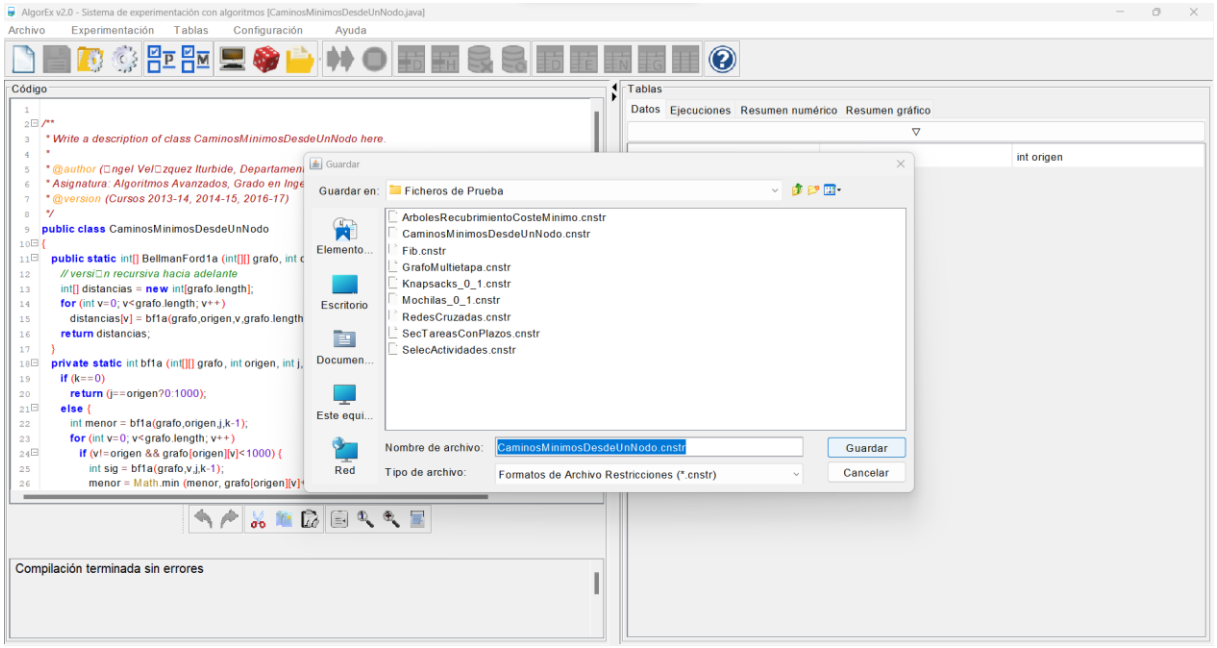


Figura 11: Diálogo de guardar restricciones (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

Tras haber guardado las restricciones en el fichero correspondiente, se nos mostrará un mensaje de éxito en el diálogo en el que estábamos.

Una vez guardadas las restricciones, aceptamos para generar los juegos de datos solicitados en base a las restricciones impuestas en el diálogo del generador (véase Figura 12).

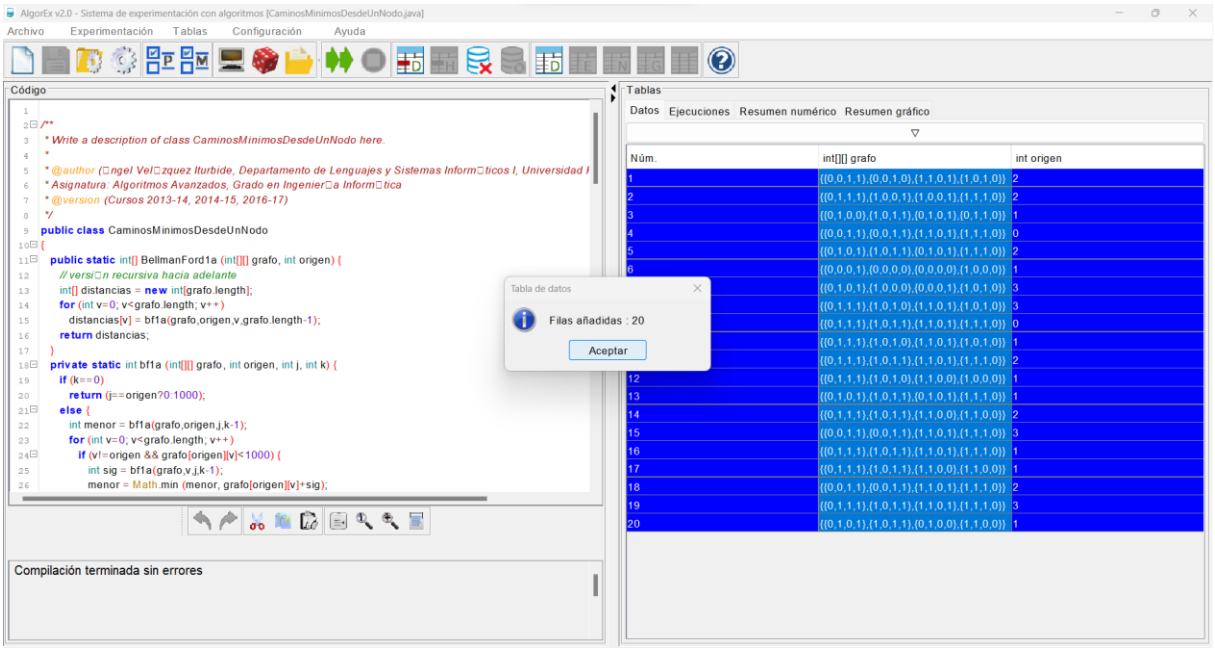


Figura 12: Resultado de la generación de juegos de datos (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

Tras haber generado los juegos de datos, los ejecutamos, obteniendo los siguientes resultados (véase Figura 13).

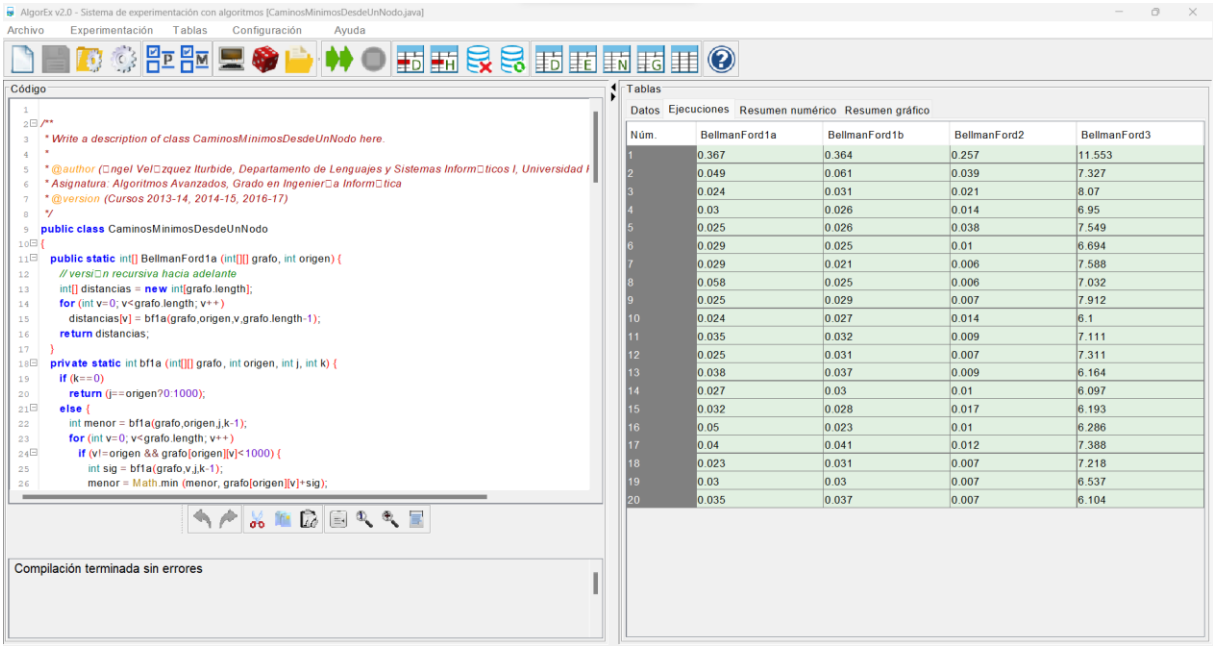


Figura 13: Tabla de ejecuciones (Caso de Uso - Generador de Generador de Grafos por Probabilidad de Arco)

Como se puede ver, en este caso, los resultados son diferentes, pues lo que se mide es la eficiencia en tiempo y no la optimalidad en base a los resultados obtenidos.

Para comprobar que se han almacenado correctamente las restricciones, eliminamos los datos de sesión y pulsamos el botón de Generar datos aleatorios para introducir el número de juegos de datos a generar, pero en este caso, marcando la opción de Cargar restricciones automáticamente (véase Figura 14).

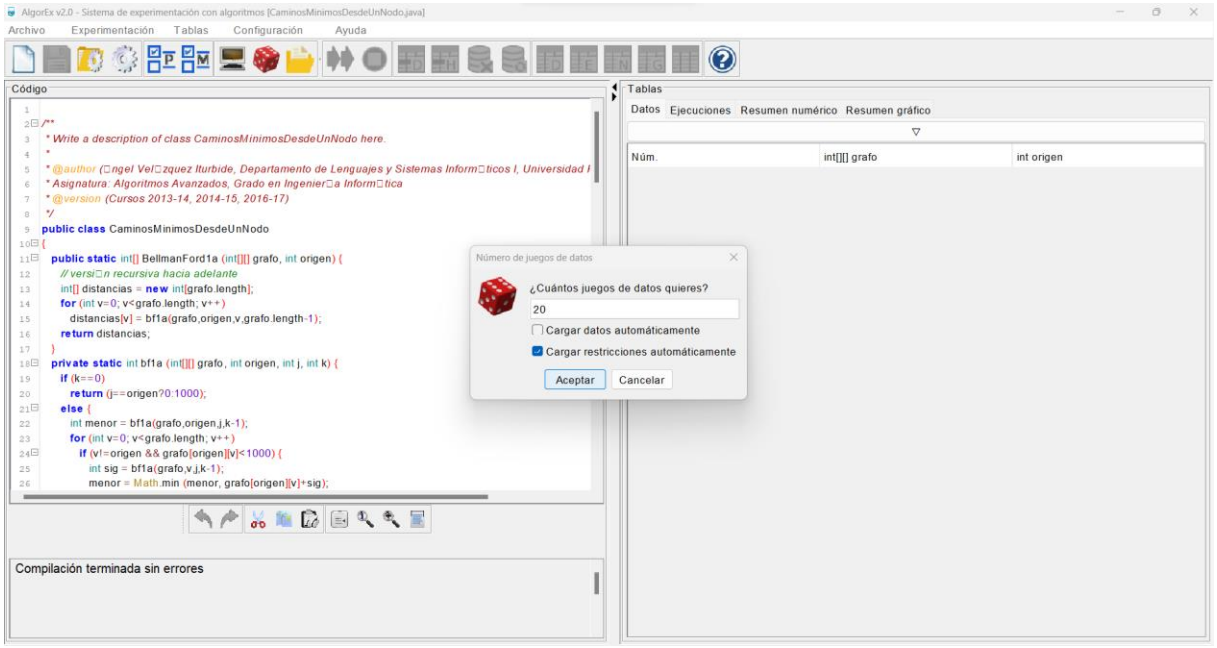


Figura 14: Cargando restricciones en el diálogo del selector de número de juegos de datos (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

A diferencia del caso anterior, como hemos marcado la opción de cargar restricciones, en este caso entramos al diálogo de Restricciones de datos, correspondiente al generador de datos principal para la carga de restricciones.

Para ver si las restricciones se han cargado correctamente, elegimos la opción de Grafo Probs. para acceder a la ventana de dicho generador (véase Figura 15).

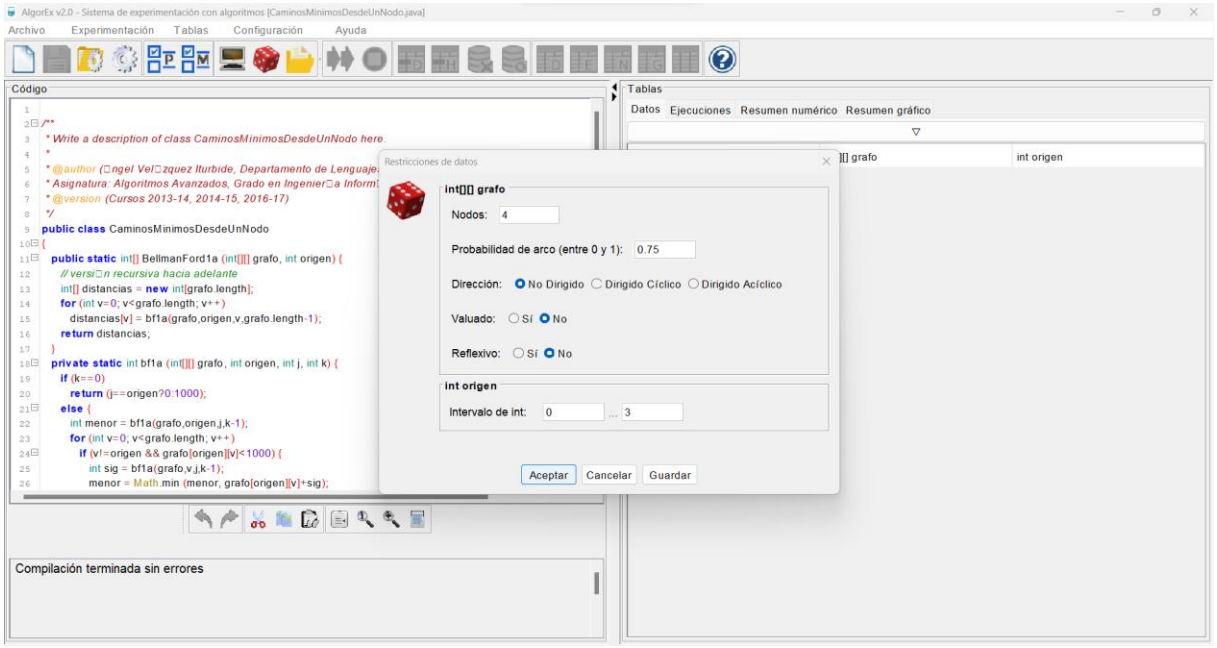


Figura 15: Diálogo del generador de grafos por probabilidad de arco tras cargar restricciones (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

Como se puede apreciar, las restricciones almacenadas anteriormente se han cargado de manera correcta, pues los valores y opciones marcadas son las mismas que cuando guardamos el fichero de restricciones: número de nodos igual a 4, la probabilidad de arco igual a 0.75, grafo no dirigido, no valuado y no reflexivo, y el entero entre 0 y 3.

Por último, para comprobar que el generador funciona bien una vez cargadas las restricciones, aceptamos para generar los juegos de datos en base a las restricciones (véase Figura 16).

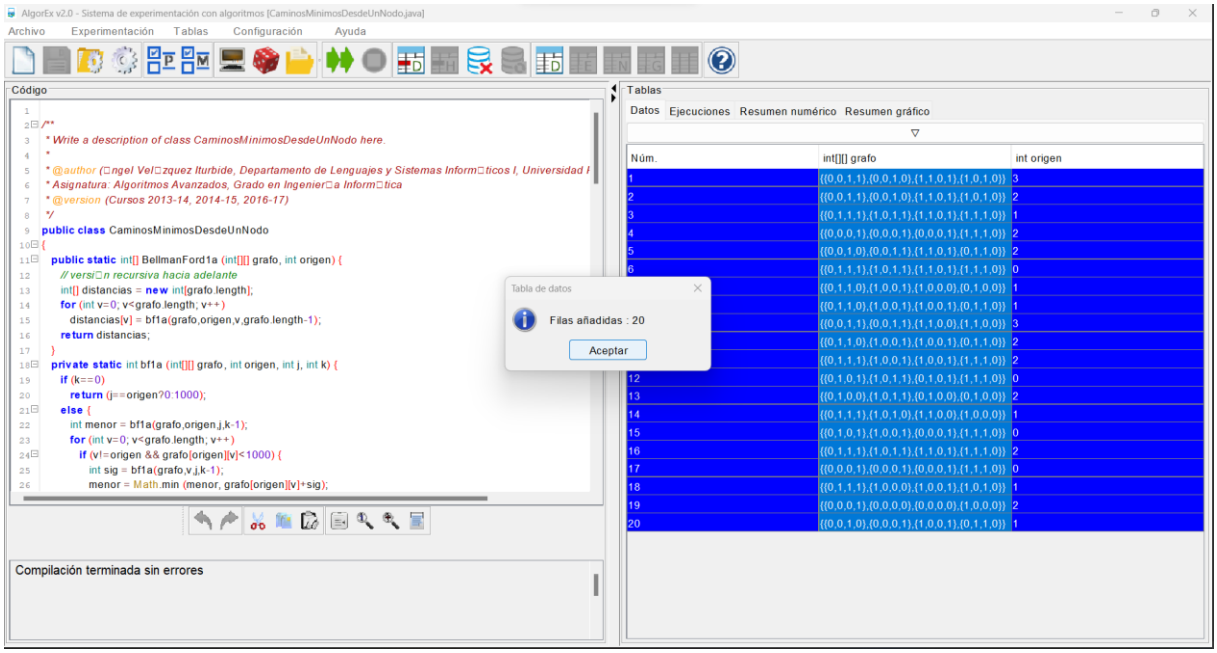


Figura 16: Resultado de la generación de juegos de datos tras cargar restricciones (Caso de Uso - Generador de Grafos por Probabilidad de Arco)

3. Metodología

En este apartado de la memoria se incluye todo lo referente a la metodología seguida para el desarrollo del proyecto: la especificación detallada de los objetivos del trabajo, el proceso de desarrollo seguido y las herramientas de desarrollo utilizadas para la elaboración del proyecto.

3.1. Especificación

Pese a que en anteriores apartados se han explicado los diferentes objetivos de este trabajo de fin de grado de una manera resumida, en este apartado de la memoria se explican todos los elementos de la especificación de este trabajo de fin de grado de forma más completa y detallada.

Así pues, los elementos incluidos en la especificación para la realización de este trabajo de fin de grado son los siguientes (en el orden de la especificación original):

- Comprobar y hacer que AlgorEx funciona con diferentes versiones de JDK (Java Development Kit), desde la versión 6 en adelante (funciona hasta la versión 19).
    - **Motivación:** en la versión anterior de AlgorEx, surgían problemas con determinadas versiones de JDK (sobre todo las más actuales).
- Ordenación de las tablas de datos y de ejecuciones. En este caso, el objetivo es que, si se ordena el contenido de alguna de las tablas por cualquiera de sus columnas, dicha reordenación se debe reproducir en la otra tabla y viceversa.
- En referencia a este aspecto, también se tenía por objetivo hacer que se pudiese reordenar la tabla de ejecuciones por cualquiera de sus columnas, puesto que en la versión anterior de AlgorEx no era posible.
    - **Motivación:** en este caso, la ordenación por columnas permite ordenar las ejecuciones en función de los resultados obtenidos o en función de los tiempos de ejecución empleados para realizar dichas ejecuciones, todo ello dependiendo del criterio de experimentación activo (optimalidad o eficiencia en tiempo).
- Para la generación aleatoria de datos, se proponían las siguientes mejoras:
    - En caso de generar datos de tipo *char*, se solicitaba incluir la posibilidad de que el generador principal admita todos los formatos válidos en Java, incluyendo caracteres Unicode (por ejemplo: ‘a’, ‘\\’, ‘\u03a9’).
        - **Motivación:** en la versión anterior de AlgorEx sólo admitía los caracteres de manera literal, es decir, sin las comillas simples, lo cual no sigue la manera habitual de insertarlos en Java.

- Generar los *n* juegos de datos indicados por el usuario, siendo todos diferentes entre sí y sin repeticiones.

  - **Motivación:** el problema que tenía la versión anterior de AlgorEx era que, en algunas ocasiones, a la hora de generar juegos de datos aleatorios, se generaban menos juegos de datos que los indicados por el usuario.

    Esto se producía en el momento en el que se generaba un juego de datos repetido, es decir, que ya estuviese en la tabla de datos.

    Por lo tanto, como no trataba de generarse otro juego de datos debido a dicha repetición, saltaba al siguiente caso y por ello se generaban menos juegos de datos.

- Reubicación del botón de "Guardar" en las ventanas de generación de datos, situándolo en la parte inferior del diálogo, junto a los botones de "Aceptar" y "Cancelar".

- Inclusión de nuevos generadores de datos, aparte del ya existente. Dichos generadores de datos son los siguientes:

  - **Generador de tareas de duración no unitaria:** este generador estará disponible en el caso de probar una función que tenga entre sus parámetros dos vectores de enteros (en Java, int[]).

    En ese caso, uno de los vectores será para los instantes de inicio (*cs*) y otro para los instantes de fin (*fs*), de tal forma que se irán generando de manera sucesiva pares de números enteros, almacenando el menor en el array *cs* y el mayor en el array *fs*.

  - **Generador de permutaciones:** este generador estará disponible en el caso de probar una función que tenga entre sus parámetros, al menos, un vector de enteros (en Java, int[]).

    En ese caso, se debe poder generar una permutación, es decir, un vector de longitud *n*, el cual contiene todos los elementos de 0 a *n*-1.

  - **Generadores de grafos:** estos generadores estarán disponibles en el caso de probar una función que tenga entre sus parámetros, al menos, una matriz de enteros (en Java, int[][]).

    En ese caso, se debe poder generar la matriz como si fuera la matriz de adyacencia de un grafo.

    El objetivo, en este caso, es crear tres generadores de grafos, los cuales son:

- **<u>Generador de grafos en base a la probabilidad de arco:</u>** en este caso, el grafo se debe generar en base a los siguientes parámetros o restricciones:

  - El número de nodos (debe ser un número natural mayor que 0).

  - La probabilidad de arco (debe ser un número real del intervalo [0…1]).

    - Este valor representa la posibilidad de que exista una conexión entre dos nodos en el grafo.

    - Con probabilidad 0 nunca se genera el arco, mientras que con probabilidad 1 se genera siempre.

  - La opción de dirección del grafo, de tal forma que puede ser:

    - **<u>No dirigido:</u>** la conexión entre los nodos es bidireccional y simétrica (si hay conexión entre el nodo A y el nodo B, también hay conexión entre el nodo B y el nodo A).

    - **<u>Dirigido cíclico:</u>** la conexión entre los nodos es unidireccional y pueden formar ciclos. Puede haber conexión del nodo A al nodo B, pero no necesariamente desde el nodo B al A, lo que permite la existencia de rutas o ciclos direccionales en el grafo.

    - **<u>Dirigido acíclico:</u>** la conexión entre los nodos es unidireccional y no pueden formar ciclos. Puede haber conexión del nodo A al nodo B, pero no desde el nodo B al A, lo que implica que no puede haber una ruta que regrese al mismo nodo de partida.

  - La opción de valuado, pues el grafo puede ser valuado (o no).

  En caso de que el grafo sea valuado, el usuario podrá indicar el rango de valores entre los que estarán los valores asignados a los arcos.

  - Además, para el caso en el que el grafo sea valuado, el usuario también podrá elegir el valor de los arcos no existentes. Todo ello mediante una opción denominada como “Infinito en grafos”, siendo por defecto Short.MAX_VALUE el valor.

  Si el grafo es no valuado, el usuario podrá indicar si el grafo es reflexivo (o no).

- **Generador de grafos en base al número de arcos:** este generador es prácticamente idéntico al anterior, con la diferencia de que la segunda de las restricciones, en lugar de ser la probabilidad de arco, es el número de arcos del grafo. Dicho valor debe estar entre 0 y n^2, aunque dependerá del caso.

- **Generador de grafos multietapa:** este tipo de grafos son dirigidos, acíclicos y valuados, de tal forma que existe un nodo origen y un nodo destino únicos. Los nodos se agrupan en etapas, y cada nodo en una etapa determinada solo tiene arcos que llegan desde nodos en la etapa anterior y salen hacia nodos de la etapa siguiente. El nodo origen tiene el índice 0, el nodo destino tiene el índice más alto n-1, y los nodos se numeran secuencialmente por etapa.

  Por lo tanto, el grafo se debe generar en base a los siguientes parámetros o restricciones:

  - El número de nodos (debe ser un número natural mayor que 0).

  - El número de etapas (debe ser un número entero del intervalo [2…n]).

  - Como el grafo es valuado, el usuario podrá indicar el rango de valores entre los que estarán los valores asignados a los arcos.

  - Además, al ser un grafo valuado, el usuario también podrá elegir el valor de los arcos no existentes. Todo ello mediante una opción denominada “Infinito en grafos”, siendo por defecto Short.MAX_VALUE.

3.2. Proceso de desarrollo

Para el desarrollo de este trabajo de fin de grado, el proceso seguido ha sido el siguiente:

- Tras la importación del proyecto original a mi entorno de desarrollo (en este caso, IntelliJ IDEA), lo primero que hice fue crearme un repositorio privado en GitHub con el contenido del proyecto, para así poder tener un control de versiones gestionado por Git.

- En base a la especificación proporcionada por el tutor de este trabajo de fin de grado, fui siguiendo en orden secuencial los objetivos que en ella se especificaban, de manera similar a lo que se conoce en el desarrollo de software como metodología en cascada.

- Por lo tanto, las etapas o pasos seguidos para el desarrollo han sido las siguientes:

  1. Funcionamiento del sistema AlgorEx con diferentes versiones de JDK.

2. Desarrollo de la reordenación de las tablas de datos y de ejecuciones.

3. Desarrollo de los cambios relativos al generador de datos original: cambios en la generación de datos de tipo *char* y el problema de la generación de juegos de datos, ya que no se generaban todos los requeridos por el usuario.

4. Desarrollo del cambio de ubicación del botón de guardar restricciones.

5. Desarrollo de manera secuencial (según el orden de la especificación) de los nuevos generadores de datos (incluyendo el tratamiento de restricciones):

    - Generador de tareas de duración no unitaria.

    - Generador de permutaciones.

    - Generador de grafos por probabilidad de arco.

    - Generador de grafos por el número de arcos.

    - Generador de grafos multietapa.

- En cada una de estas etapas, tras cada avance realizado, me he comunicado con el tutor por correo electrónico, mandándole las sucesivas versiones del proyecto con dichos avances.

    - Además, para todas las dudas que me han ido surgiendo en el desarrollo se las he comentado al tutor en sendos correos electrónicos y algunas reuniones.

Como se puede observar, se ha aplicado un enfoque personalizado basado en elementos de metodologías reconocidas en el desarrollo de software. Pese a no adherirme de manera estricta a una metodología concreta, se han adoptado prácticas que han permitido un progreso ordenado y una comunicación efectiva con el tutor.

La secuencia de objetivos seguida en el desarrollo ha sido en orden secuencial, adoptando una estructura similar a la metodología en cascada. Mediante ello, se pudo abordar cada objetivo de manera sistemática, avanzando al siguiente tras haber completado el anterior, asegurando una gestión eficiente del progreso del proyecto.

A pesar de no ser mencionado explícitamente en el proceso, también se ha adoptado un enfoque iterativo, estableciendo una comunicación continua con el tutor a través de diferentes correos electrónicos y de reuniones. Esta iteración constante permitió que el trabajo se ajustase de mejor manera a los elementos de la especificación aportada por el tutor.

Otro aspecto fundamental ha sido la implementación de un sistema de control de versiones mediante un repositorio en GitHub, utilizando la tecnología Git. Esto ha permitido un seguimiento preciso de los cambios realizados, una gestión efectiva de versiones y una colaboración más eficiente con el tutor.

En resumen, se ha realizado un enfoque personalizado en el que se han incorporado elementos de diferentes metodologías reconocidas en el desarrollo del software y, pese no haberme adherido a una metodología específica, las prácticas aplicadas han garantizado un progreso eficiente, siempre centrado en el usuario.

3.3. Herramientas de desarrollo

En este apartado de la memoria se incluyen todas las herramientas utilizadas para el desarrollo del proyecto de este trabajo de fin de grado.

- Para el control de versiones, he utilizado Git y GitHub.

- Para el desarrollo del código y las pruebas he utilizado, como entorno de desarrollo, el programa IntelliJ IDEA, en su edición Community (durante el desarrollo) y su edición Ultimate (para la generación de diagramas UML).

- Para la compilación y ejecución de las versiones del sistema, he utilizado la versión 19 de JDK, pues era la última versión disponible cuando comencé con el proyecto.

- Asimismo, se han usado algunas herramientas para otras tareas del trabajo de fin de grado distintas del desarrollo de software:

    - Para las búsquedas en Internet, con el fin de encontrar referencias para solventar los problemas con los que me he encontrado, he utilizado como navegador Mozilla Firefox.

    - Para redactar esta memoria, he utilizado Microsoft Word, en la versión de Office 365.

    - Para la edición de algunas de las figuras incluidas en el apartado de *Diseño* he utilizado Paint 3D, de Microsoft.

    - Para la comunicación mediante correo electrónico con el tutor he utilizado Outlook, de Microsoft, como herramienta de correo electrónico.

    - Como herramienta para videoconferencias para las reuniones con el tutor he utilizado Microsoft Teams, en su versión educativa.

4. Diseño

En este apartado de la memoria se incluye todo lo referente a la interfaz de usuario (sobre todo de las novedades introducidas) y de la arquitectura de las clases y paquetes utilizados y modificados en el proyecto.

4.1. Interfaz de usuario

Dentro de las mejoras desarrolladas en base a los objetivos incluidos en la especificación, se han realizado diferentes cambios en algunos de los diálogos y tablas de la interfaz, además de la creación de nuevos diálogos para los nuevos generadores de datos desarrollados.

Por lo tanto, en este apartado se irán explicando paso por paso los cambios que se han ido incluyendo sobre el sistema AlgorEx.

4.1.1. Ordenación de las tablas de datos y ejecuciones

Uno de los primeros cambios en la interfaz de usuario ha sido el hecho de poder ordenar la tabla de ejecuciones por cualquiera de sus columnas.

- Para ello, se estableció un ordenador de filas (conocido en Java como RowSorter) al panel correspondiente, todo ello en la clase en la que se renderiza dicho panel (Modelo2RenderaColor.java → Se explicará con más detalle más adelante, en el apartado de Implementación).

Para poder ilustrar el funcionamiento de la reordenación en la tabla de ejecuciones, se prueba a modo de ejemplo la clase Fib.java. Esta clase dispone de diferentes algoritmos para obtener el resultado de la aplicación de la sucesión de Fibonacci de un número en concreto.

En dicho ejemplo, se han generado 20 juegos de datos con enteros aleatorios en un rango de 0 a 20 y, tras ello, se han ejecutado las diferentes funciones por cada uno de esos juegos de datos.

Una vez dentro de la tabla de ejecuciones, se ha pulsado sobre la tercera columna (correspondiente a la función *fib2*), con el fin de ordenar los resultados de la tabla de ejecuciones en función del orden ascendente de los resultados de dicha función.

En la siguiente figura (véase Figura 17), se muestra la secuencia del antes y el después de la reordenación por la columna anteriormente mencionada.

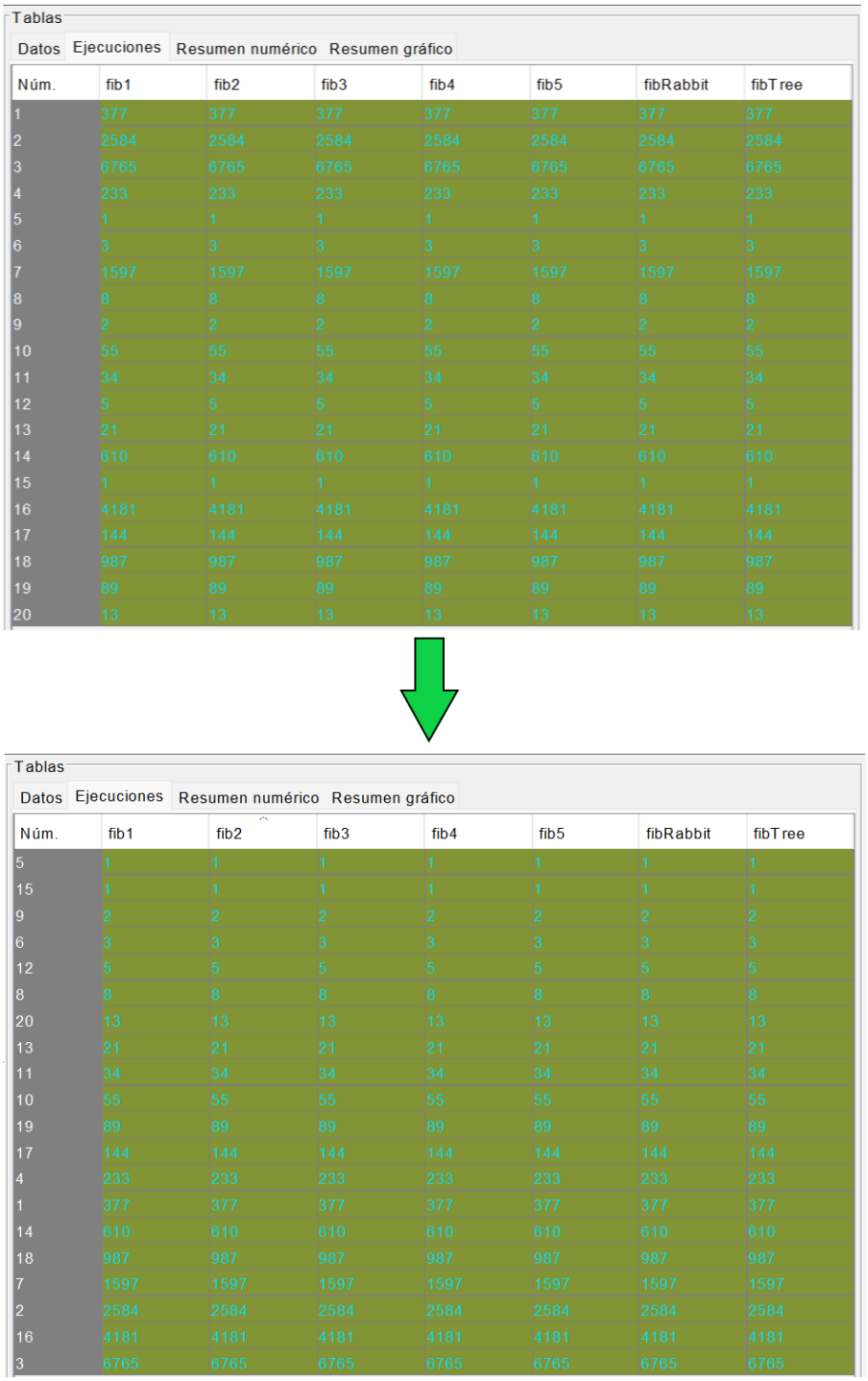


Figura 17: Reordenación de la tabla de ejecuciones por la tercera columna (ascendente)

Otro de los cambios en referencia a la ordenación de las tablas fue el hecho de que la ordenación de las tablas de datos y de ejecuciones estuviese sincronizada, es decir, que al ordenar cualquiera de las tablas por cualquiera de sus columnas, la ordenación se acarrease de una a la otra y viceversa.

- Para ello, fue necesario adaptar la clase *ModeloTabla* para poder mover filas de una posición a otra, además de crear una clase que heredase de *RowSorterListener* para

poder gestionar los eventos de ordenación y así poder sincronizar la ordenación de ambas tablas → Se explicará con más detalle más adelante, en el apartado de Implementación.

Para poder ilustrar el funcionamiento de la sincronización del orden de las tablas de datos y de ejecuciones, se prueba, al igual que antes, la clase Fib.java. Esta clase dispone de diferentes algoritmos para obtener el resultado de la aplicación de la sucesión de Fibonacci de un número en concreto.

En dicho ejemplo, se han generado 20 juegos de datos con enteros aleatorios en un rango de 0 a 20 y, tras ello, se han ejecutado las diferentes funciones por cada uno de esos juegos de datos.

En las siguiente figuras (véase Figuras 18 y 19), se muestra la secuencia del antes y el después del orden de las tablas de datos y de ejecuciones, tras haber ordenado por la tercera columna (correspondiente a los resultados de la función *fib2*) en la tabla de ejecuciones.

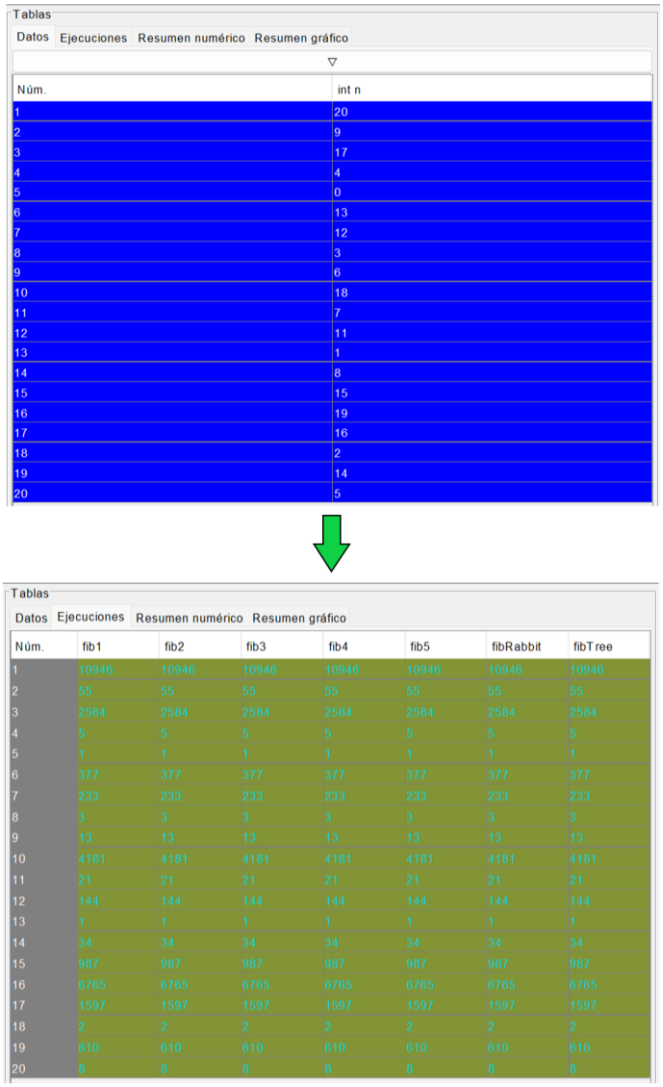


Figura 18: Proceso de sincronización de la ordenación de las tablas de datos y de ejecuciones (Estado Original)

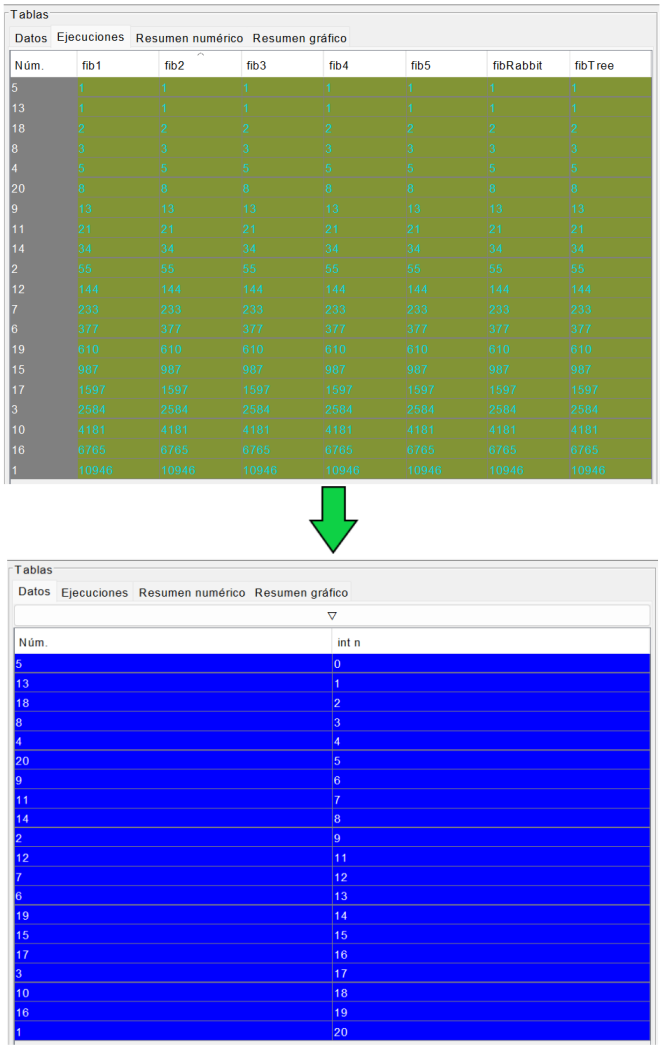


Figura 19: Proceso de sincronización de la ordenación de las tablas de datos y de ejecuciones (Estado Final)

Pese a que la sincronización del orden de las tablas funciona bien, debido a la implementación realizada para dicha sincronización (explicado en el apartado de *Implementación*) y a las limitaciones del modelo de tabla utilizado en la implementación, si queremos ordenar desde la otra tabla (no desde la que ya hemos ordenado), será necesario ordenar primero por la columna "Núm." en la tabla en la que hayamos ordenado en primer lugar, para evitar conflictos en la sincronización.

Esto se debe a la implementación de la sincronización del orden, pues se toma como referencia el orden de los elementos de la primera columna de ambas tablas (que es la columna "Núm.").

4.1.2. Cambio de ubicación del botón de Guardar

Otro de los cambios en la interfaz de usuario ha sido el hecho de cambiar la posición del botón de "Guardar", a través del cual es posible almacenar en un fichero de extensión .cnstr las restricciones establecidas por el usuario.

- En la versión anterior de AlgorEx este botón aparecía bajo las restricciones a introducir por el usuario.

- Por ello, se solicitaba cambiarlo de sitio y que estuviese situado en la parte inferior del diálogo del generador de datos, a la derecha de los botones de “Aceptar” y “Cancelar”, ya que, en ocasiones, dependiendo del tamaño del diálogo, no se veía correctamente el botón.

A continuación, en la siguiente imagen, se muestra cómo estaba el botón en la versión anterior y cómo se ve actualmente (véase Figura 20).

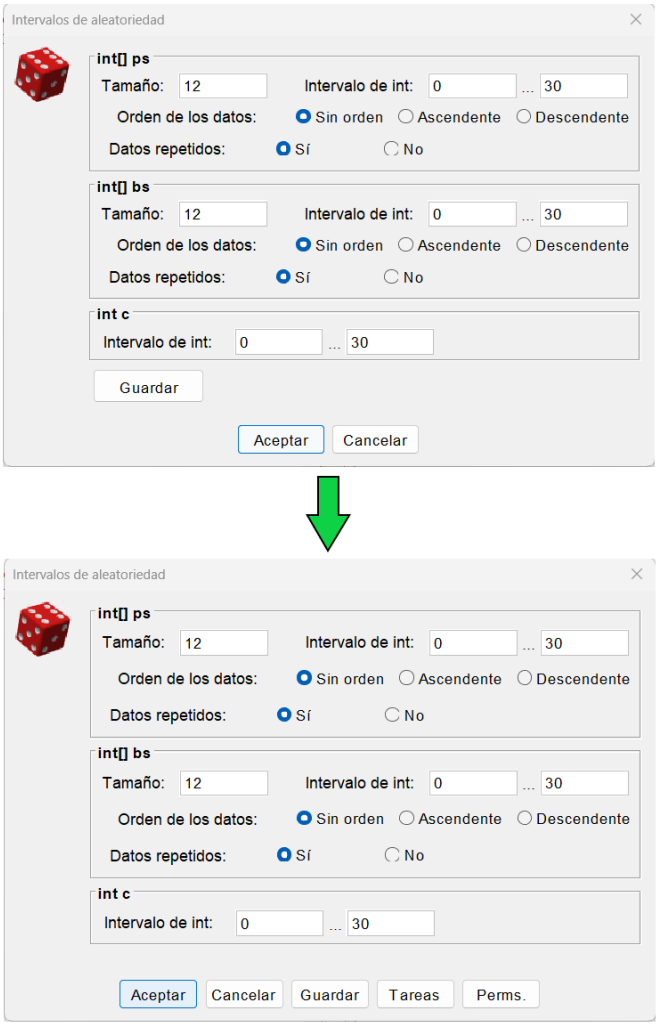


Figura 20: Cambio de posición del botón de "Guardar"

4.1.3. Diálogo del generador de tareas

Para poder acceder al diálogo de este generador, será necesario probar una función que contenga dos vectores de enteros como parámetros.

Así, en el diálogo del generador de datos principal aparecerá un botón llamado “Tareas”, mediante el cual podemos acceder al diálogo de este nuevo generador.

En este caso, el diálogo se compone de un pequeño panel con dos restricciones:

1. El tamaño de los vectores a generar.

2. El rango de valores que se irán insertando en los vectores, dividiéndose en dos campos: uno para el valor mínimo del rango, y otro para el valor máximo del rango.

Además, si la función a probar incluye más parámetros, los paneles correspondientes a dichos parámetros tendrán la misma forma que en el diálogo del generador de datos original.

En la siguiente imagen se muestra la secuencia para el acceso al diálogo correspondiente a este generador de datos (véase Figura 21).

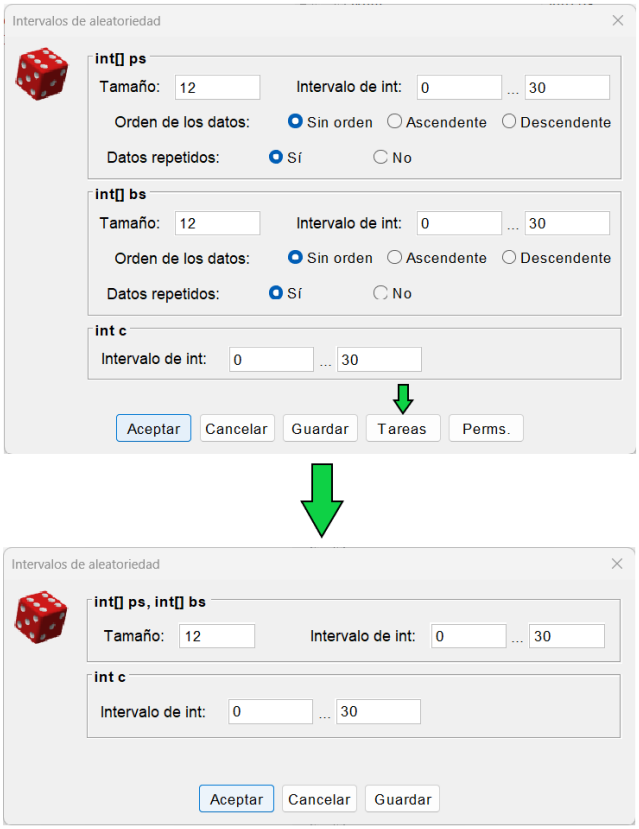


Figura 21: Secuencia de acceso al diálogo del generador de tareas de duración no unitaria

En el caso en que se introduzca un tamaño para los vectores que sea menor que 0, se muestra el siguiente mensaje de error (véase Figura 22).

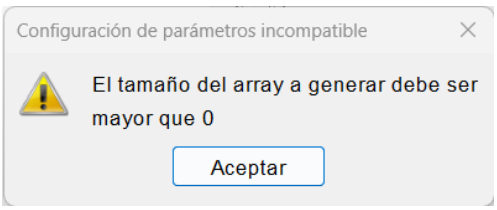


Figura 22: Mensaje de error en caso de que el tamaño introducido de los vectores sea menor que 0

4.1.4. Diálogo del generador de permutaciones

Para poder acceder al diálogo de este generador, será necesario probar una función que contenga, al menos, un vector de enteros entre sus parámetros.

Así, en el diálogo del generador de datos principal aparecerá un botón llamado “Perms.”, mediante el cual podemos acceder al diálogo de este nuevo generador.

En este caso, el diálogo se compone de un pequeño panel con una sola restricción: el tamaño del vector a generar.

Además, si la función a probar incluye más parámetros, los paneles correspondientes a dichos parámetros tendrán la misma forma que en el diálogo del generador de datos original (salvo en el caso en que se trate de un vector de enteros, pues será igual que el mencionado anteriormente).

En la siguiente imagen se muestra la secuencia para el acceso al diálogo correspondiente a este generador de datos (véase Figura 23).

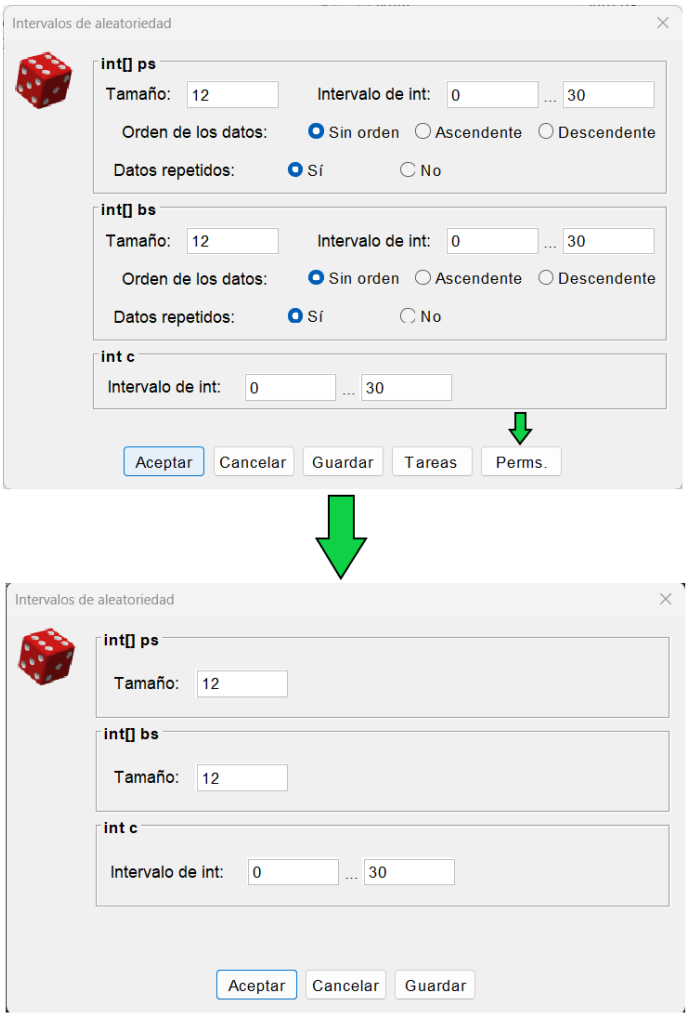


Figura 23: Secuencia de acceso al diálogo del generador de permutaciones

Para este generador de datos, dependiendo del caso, existen dos mensajes de error diferentes:

- El primero de ellos aparece si el tamaño establecido para el vector es menor que 0, como se puede ver en la *Figura 21*.

- El segundo de ellos aparece en el caso en que el número de juegos de datos sea mayor que el número de permutaciones posibles dado el tamaño del vector (véase Figura 24).

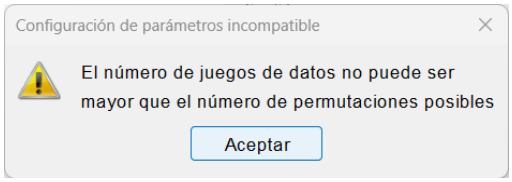


Figura 24: Mensaje de error en caso de que el número de juegos de datos sea mayor al número de permutaciones posibles (Generador de Permutaciones)

4.1.5. Diálogo del generador de grafos por probabilidad de arco

Para poder acceder al diálogo de este generador, será necesario probar una función que contenga, al menos, una matriz de enteros entre sus parámetros, ya que, dependiendo del caso, puede tratarse de la matriz de adyacencia de un grafo.

Así, en el diálogo del generador de datos principal aparecerá un botón llamado “Grafo Probs.” (junto a los botones “Grafo Arcos” y “Grafo Etapas”, evidentemente), mediante el cual podemos acceder al diálogo de este nuevo generador.

En este caso, el diálogo se compone, en primera instancia, de un panel con cuatro restricciones:

1. El número de nodos, el cual debe ser un número natural mayor que 0.

2. La probabilidad de arco, la cual debe ser un número real del intervalo [0…1].

3. Un selector que nos permite indicar la dirección del grafo. El grafo puede ser no dirigido, dirigido cíclico o dirigido acíclico.

4. Un selector que nos permite indicar si el grafo es valuado o no. Dependiendo de la opción seleccionada, se incluirán unas restricciones u otras en el panel:

    - Si se selecciona que sí es valuado, aparecerán dos nuevas restricciones:

        1. El rango de valores que tendrán los arcos existentes, comenzando por 1.

        2. Un selector, mediante el cual se puede elegir si es Infinito en grafos o no:

- ❖ Si se elige que sí, se mostrará otra restricción en la que se podrá elegir el valor para los arcos no existentes.

- ❖ Si se elige que no, se tomará como valor para los arcos no existentes el valor de Short.MAX_VALUE.

- o Si se selecciona que no es valuado, aparecerá una nueva restricción: un selector que nos permite indicar si el grafo es reflexivo o no.

Además, si la función a probar incluye más parámetros, los paneles correspondientes a dichos parámetros tendrán la misma forma que en el diálogo del generador de datos original (salvo en el caso en que se trate de una matriz de enteros, pues será igual que el mencionado anteriormente).

En la siguiente imagen se muestra la secuencia para el acceso al diálogo correspondiente a este generador de datos (véase Figura 25).

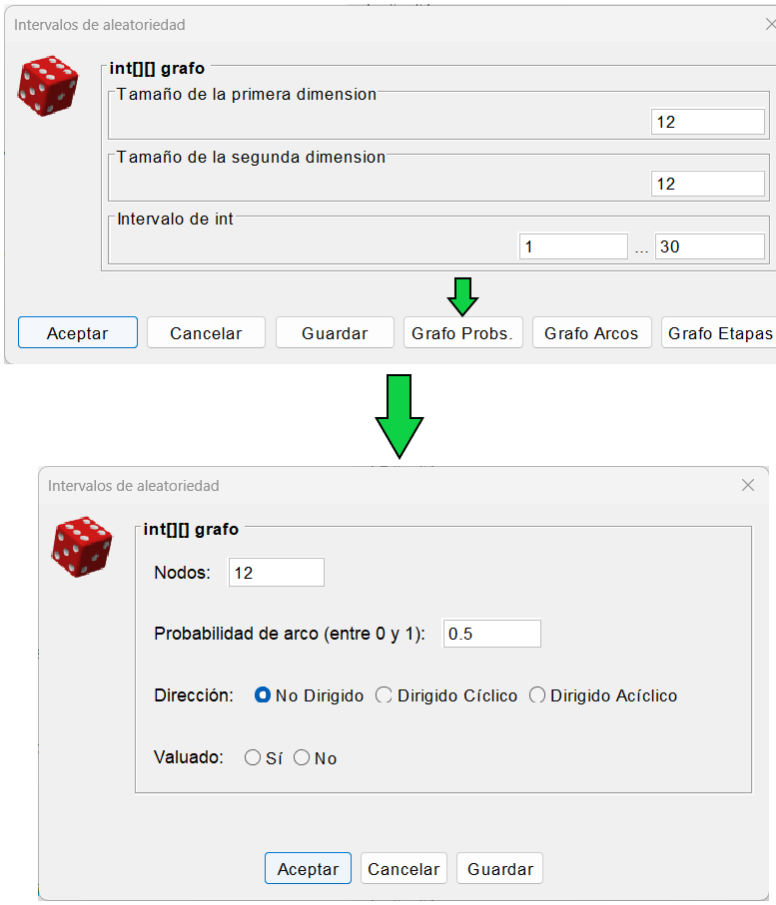
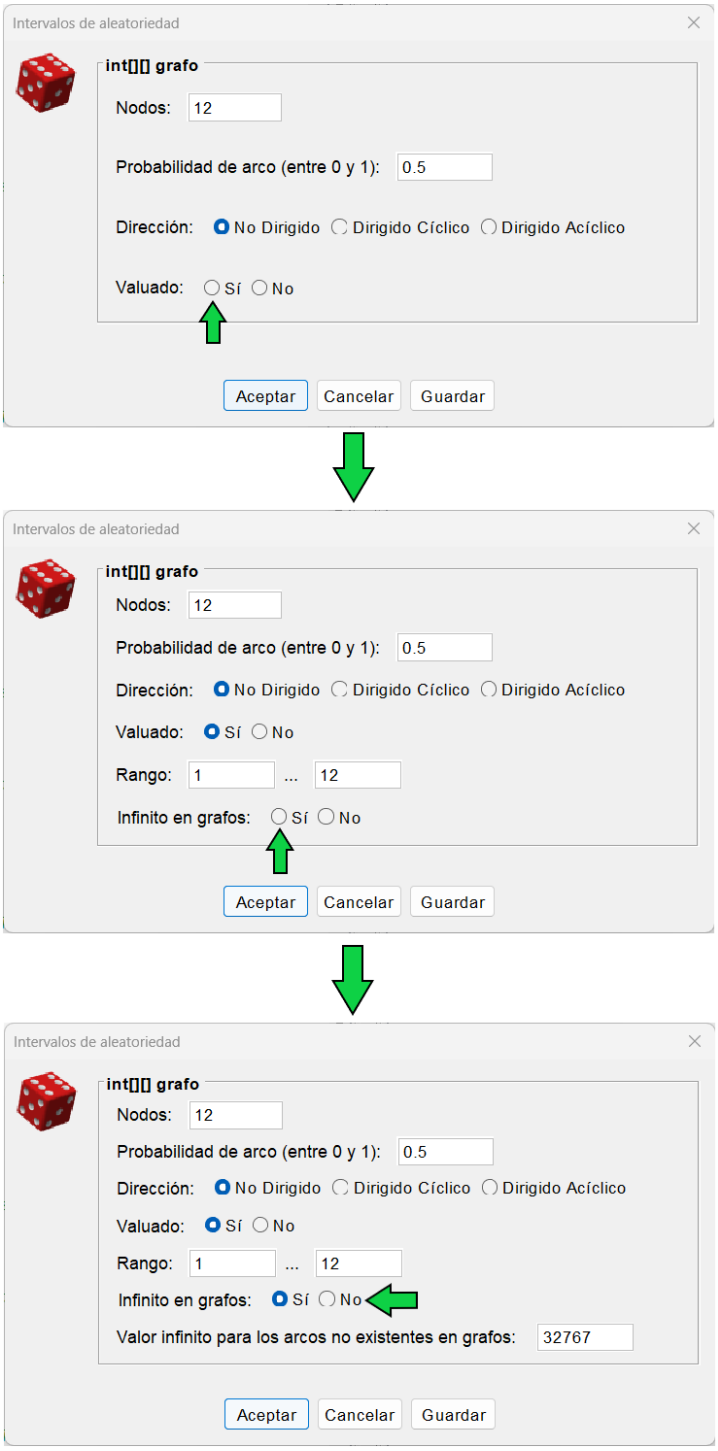


Figura 25: Secuencia de acceso al diálogo del generador de grafos por probabilidad de arco

Como se puede apreciar, por defecto, el número de nodos es igual a 12, la probabilidad de arco es igual a 0.5 y en el selector de la dirección del grafo está marcada la opción de no dirigido.

Sin embargo, en el selector que marca si el grafo es valuado o no, no está marcada ninguna opción. Esto es debido a que, como bien se ha comentado con anterioridad, dependiendo de la opción elegida, se muestran unas restricciones u otras.

En la siguiente imagen se muestra la secuencia de posibles restricciones que se muestran si pulsamos el botón “Sí” en el selector “Valuado” (véase Figura 26).



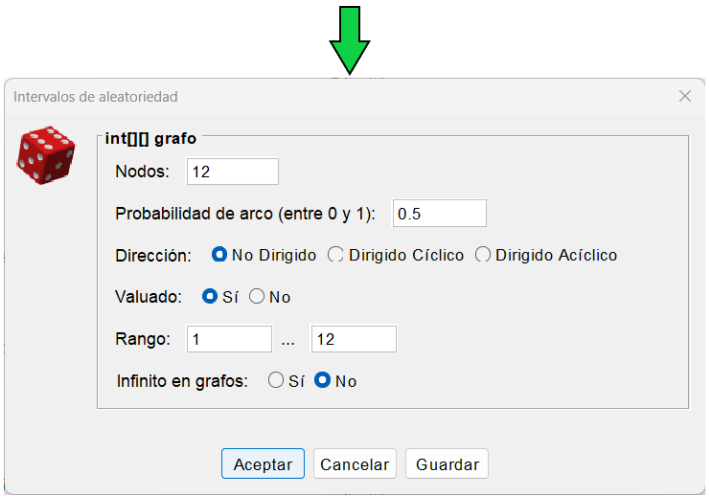


Figura 26: Secuencia de posibles restricciones en el diálogo del generador de grafos por probabilidad de arco (en caso de escoger que sea valuado)

En el caso de pulsar el botón "No" en el selector "Valuado", la secuencia será la siguiente, partiendo del último estado de la imagen anterior (véase Figura 27).

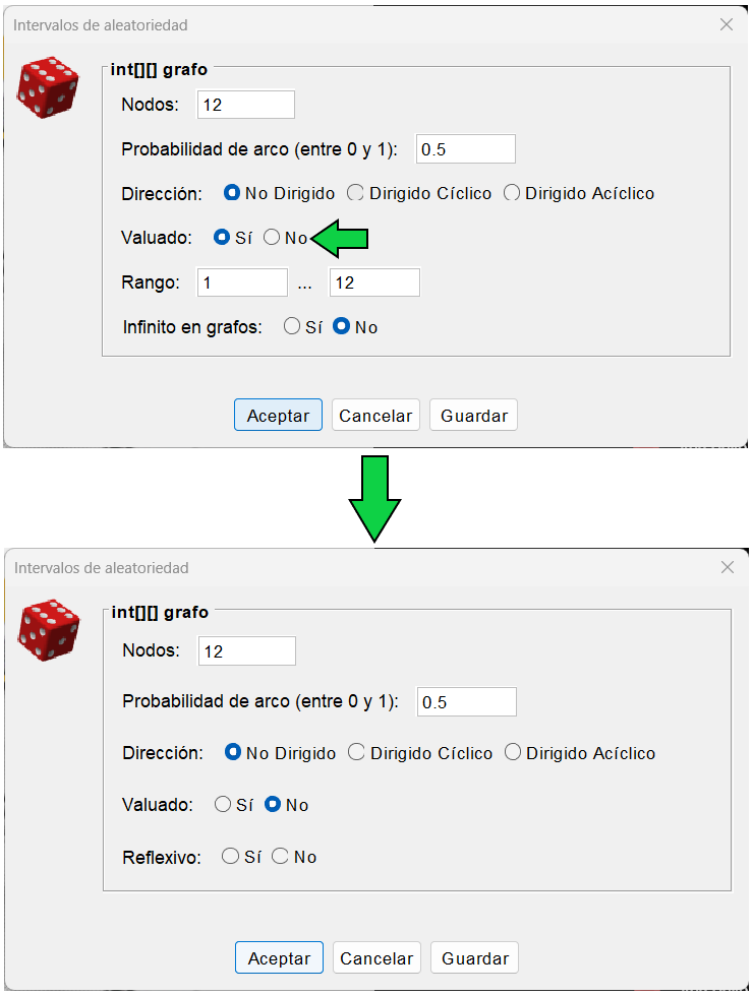


Figura 27: Secuencia de posibles restricciones en el diálogo del generador de grafos por probabilidad de arco (en caso de escoger que no sea valuado)

Para este generador de datos, dependiendo de los valores introducidos o por la falta de opciones seleccionadas, se mostrarán diferentes mensajes de error:

- En el caso de introducir un número de nodos menor que 0, se mostrará el siguiente mensaje de error (véase Figura 28).

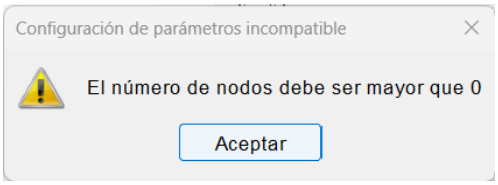


Figura 28: Mensaje de error en caso de introducir un número de nodos menor que 0

- En el caso de introducir un número real mayor que 1 o menor que 0 en el campo de probabilidad de arco, se mostrará el siguiente mensaje de error (véase Figura 29).

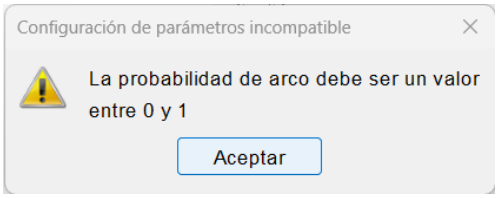


Figura 29: Mensaje de error en caso de introducir una probabilidad de arco incorrecta (Generador de Grafos por Probabilidad de Arco)

- En el caso de:
    - Dejar el campo del número de nodos vacío.
    - Dejar el campo de probabilidad de arco vacío.
    - No pulsar ninguna opción en el selector de “Valuado”.
    - Pulsar el botón “Sí” en el selector de “Valuado” y dejar, al menos, uno de los campos del rango vacío.
    - Pulsar el botón “Sí” en el selector de “Valuado” y no pulsar ningún botón en el selector de “Infinito en grafos”.
    - Pulsar el botón “Sí” en el selector de “Valuado”, pulsar el botón “Sí” en el selector de “Infinito en grafos” y dejar el campo del valor para los arcos inexistentes vacío.
    - Pulsar el botón “No” en el selector de “Valuado” y no pulsar ningún botón en el selector de “Reflexivo”.

Se mostrará el siguiente mensaje de error, indicándonos que debemos completar todos los parámetros del panel (véase Figura 35).

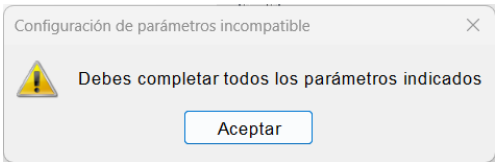


Figura 30: Mensaje de error por falta de parámetros (Generador de Grafos por Probabilidad de Arco)

4.1.6. Diálogo del generador de grafos por número de arcos

Para poder acceder al diálogo de este generador, será necesario probar una función que contenga, al menos, una matriz de enteros entre sus parámetros, ya que, dependiendo del caso, puede tratarse de la matriz de adyacencia de un grafo.

Así, en el diálogo del generador de datos principal aparecerá un botón llamado "Grafo Arcos" (junto a los botones "Grafo Probs." y "Grafo Etapas"), mediante el cual se accede al diálogo de este nuevo generador.

En este caso, el diálogo es prácticamente idéntico al del generador anterior. Se compone, en primera instancia, de un panel con cuatro restricciones, de las cuales sólo cambia la segunda:

- Se corresponde con el número de arcos, la cual debe ser un número natural entre 0 y un número *n* determinado, el cual depende de diferentes casos:
    - Si es un grafo no dirigido o un grafo dirigido acíclico, tenemos dos casos:
        - Si el grafo es reflexivo, el número máximo de arcos es igual a $n^2/2$.
        - Si el grafo es no reflexivo, el número máximo de arcos es igual a $(n^2-n)/2$.
    - Si es un grafo dirigido cíclico, tenemos dos casos:
        - Si el grafo es reflexivo, el número máximo de arcos es igual a n^2.
        - Si el grafo es no reflexivo, el número máximo de arcos es igual a $n^2/2$.

Además, si la función a probar incluye más parámetros, los paneles correspondientes a dichos parámetros tendrán la misma forma que en el diálogo del generador de datos original (salvo en el caso en que se trate de una matriz de enteros, pues será igual que el mencionado anteriormente).

En la siguiente imagen se muestra la secuencia para el acceso al diálogo correspondiente a este generador de datos (véase Figura 31).

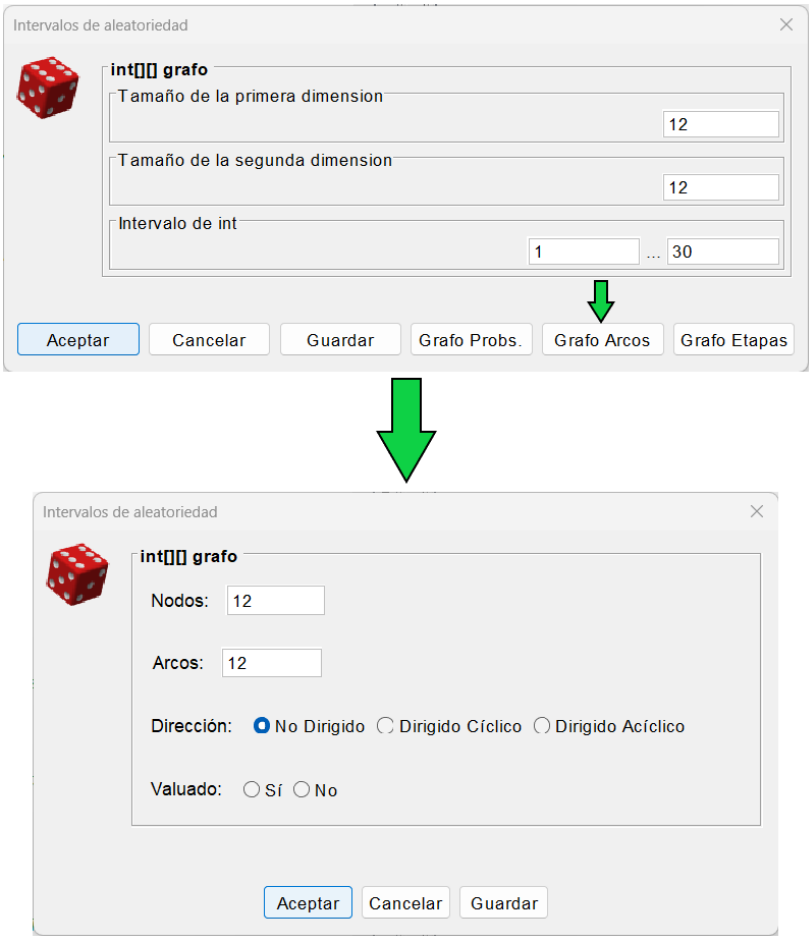
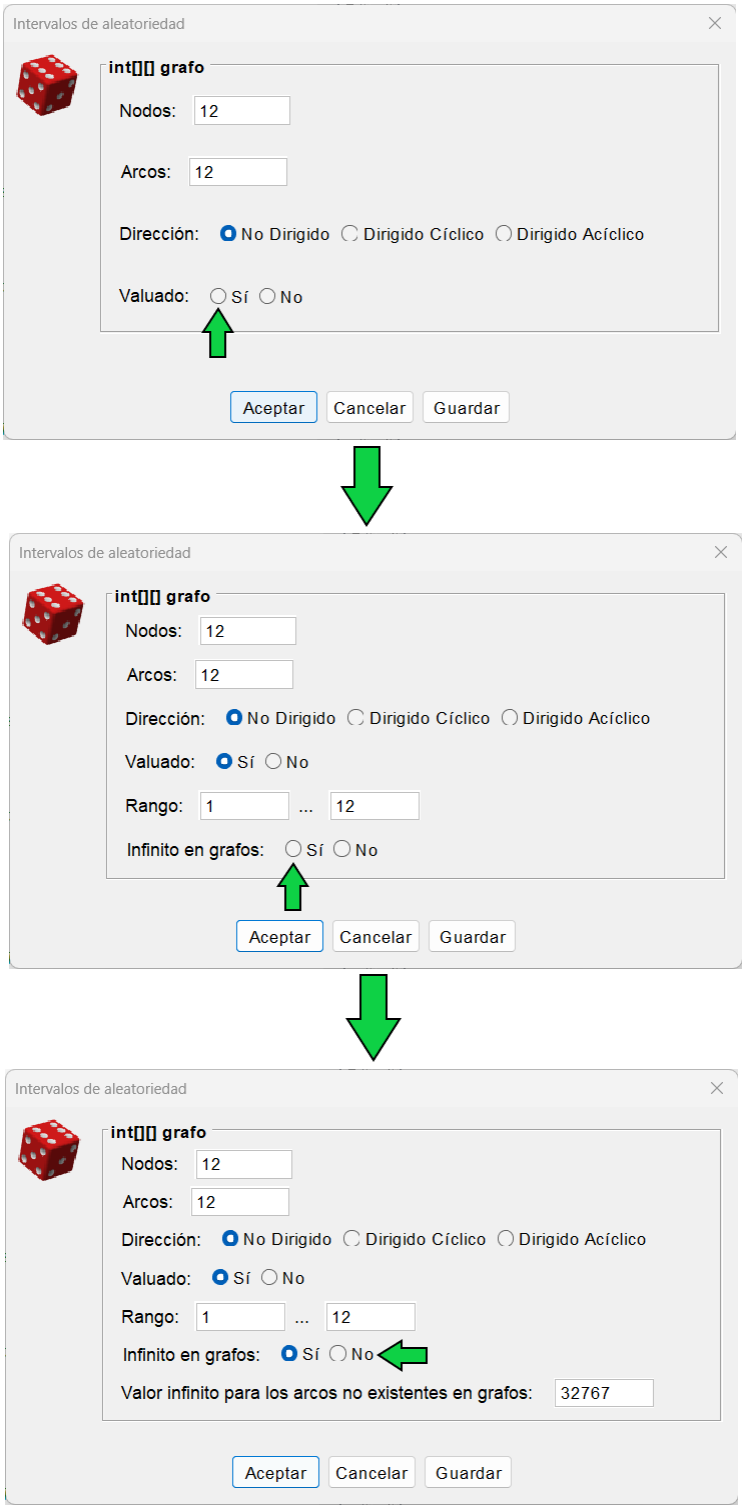


Figura 31: Secuencia de acceso al diálogo del generador de grafos por número de arcos

Como se puede apreciar, por defecto, el número de nodos es igual a 12, el número de arcos es igual a 12 y en el selector de la dirección del grafo está marcada la opción de no dirigido.

Sin embargo, en el selector que marca si el grafo es valuado o no, no está marcada ninguna opción. Esto es debido a que, como bien se ha comentado con anterioridad, dependiendo de la opción elegida, se muestran unas restricciones u otras.

En la siguiente imagen se muestra la secuencia de posibles restricciones que se muestran si pulsamos el botón “Sí” en el selector “Valuado” (véase Figura 32).


Intervalos de aleatoriedad
int[][] grafo
Nodos: 12
Arcos: 12
Dirección: No Dirigido Dirigido Cíclico Dirigido Acíclico
Valuado: Sí No
Aceptar Cancelar Guardar
Intervalos de aleatoriedad
int[][] grafo
Nodos: 12
Arcos: 12
Dirección: No Dirigido Dirigido Cíclico Dirigido Acíclico
Valuado: Sí No
Rango: 1 ... 12
Infinito en grafos: Sí No
Aceptar Cancelar Guardar
Intervalos de aleatoriedad
int[][] grafo
Nodos: 12
Arcos: 12
Dirección: No Dirigido Dirigido Cíclico Dirigido Acíclico
Valuado: Sí No
Rango: 1 ... 12
Infinito en grafos: Sí No
Valor infinito para los arcos no existentes en grafos: 32767
Aceptar Cancelar Guardar

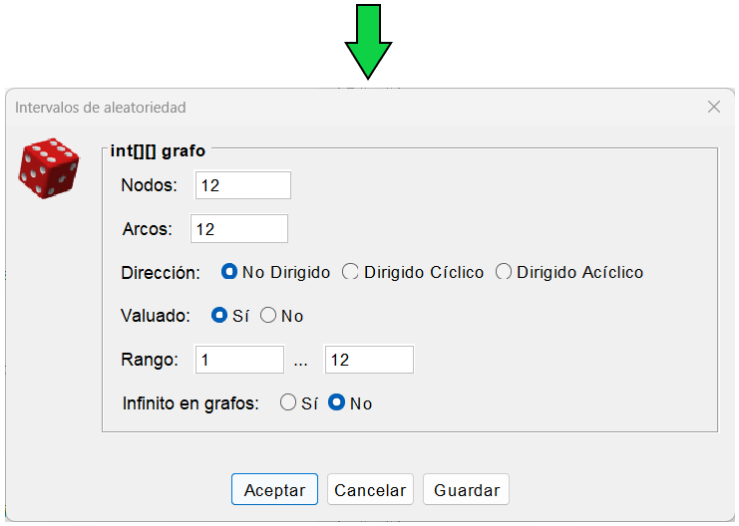


Figura 32: Secuencia de posibles restricciones en el diálogo del generador de grafos por número de arcos (en caso de escoger que sea valuado)

En el caso de pulsar el botón "No" en el selector "Valuado", la secuencia será la siguiente, partiendo del último estado de la imagen anterior (véase Figura 33).

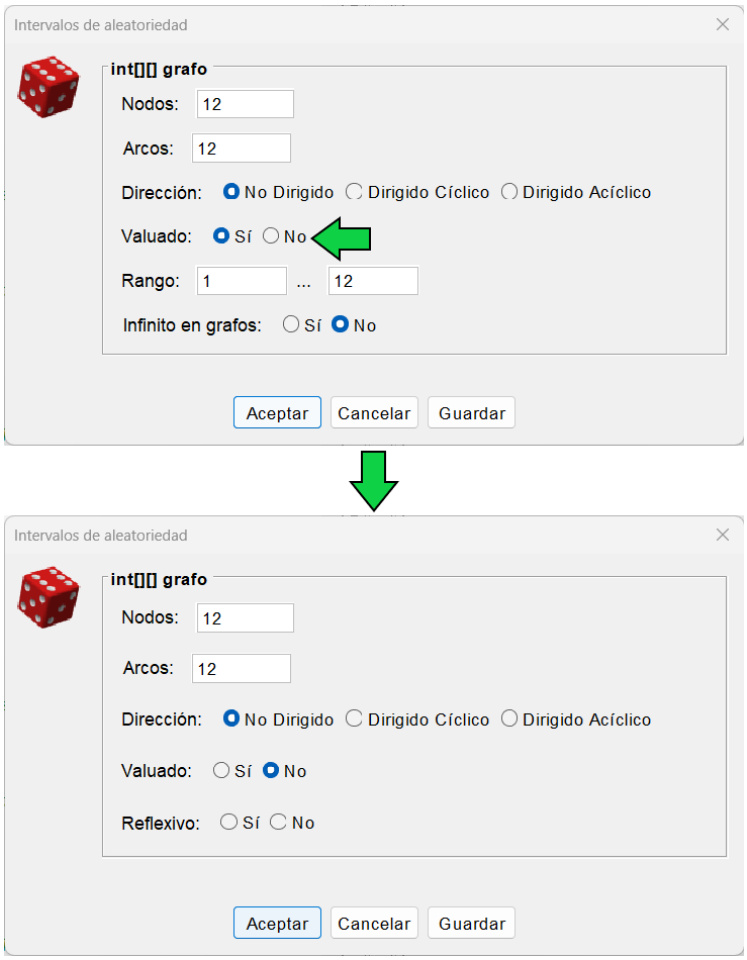


Figura 33: Secuencia de posibles restricciones en el diálogo del generador de grafos por número de arcos (en caso de escoger que no sea valuado)

Para este generador de datos, dependiendo de los valores introducidos o por la falta de opciones seleccionadas, se mostrarán diferentes mensajes de error:

- En el caso de introducir un número de nodos menor que 0, se mostrará el mensaje de error visto en la *Figura 28*.

- En el caso de introducir un número de arcos inválido (es decir, fuera de los rangos mencionados con anterioridad), se mostrará el siguiente mensaje de error (véase Figura 34).

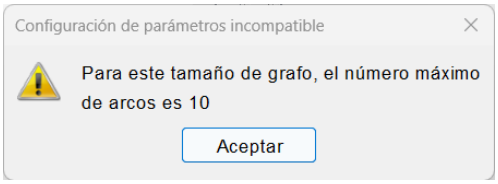


Figura 34: Mensaje de error en caso de introducir un número de arcos inválido (Generador de Grafos por Número de Arcos)

- En el caso de:

    - Dejar el campo del número de nodos vacío.

    - Dejar el campo de número de arcos vacío.

    - No pulsar ninguna opción en el selector de “Valuado”.

    - Pulsar el botón “Sí” en el selector de “Valuado” y dejar, al menos, uno de los campos del rango vacío.

    - Pulsar el botón “Sí” en el selector de “Valuado” y no pulsar ningún botón en el selector de “Infinito en grafos”.

    - Pulsar el botón “Sí” en el selector de “Valuado”, pulsar el botón “Sí” en el selector de “Infinito en grafos” y dejar el campo del valor para los arcos inexistentes vacío.

    - Pulsar el botón “No” en el selector de “Valuado” y no pulsar ningún botón en el selector de “Reflexivo”.

  Se mostrará el mensaje de error visto en la *Figura 35*, indicándonos que debemos completar todos los parámetros del panel.

4.1.7. Diálogo del generador de grafos multietapa

Para poder acceder al diálogo de este generador, será necesario probar una función que contenga, al menos, una matriz de enteros entre sus parámetros, ya que, dependiendo del caso, puede tratarse de la matriz de adyacencia de un grafo.

Así, en el diálogo del generador de datos principal aparecerá un botón llamado “Grafo Etapas” (junto a los botones “Grafo Probs.” y “Grafo Arcos”), mediante el cual se accede al diálogo de este nuevo generador.

En este caso, el diálogo se compone, en primera instancia, de un panel con cuatro restricciones:

1. El número de nodos, el cual debe ser un número natural mayor que 0.

2. El número de etapas, el cual debe ser un número natural entre 2 y el número de nodos.

3. Como el grafo multietapa es valuado, se muestran dos restricciones más:

    3.1. El rango de valores que tendrán los arcos existentes, comenzando por 1.

    3.2. Un selector, mediante el cual se puede elegir si es Infinito en grafos o no:

    - Si se elige que sí, se mostrará otra restricción en la que se podrá elegir el valor para los arcos no existentes.

    - Si se elige que no, se tomará como valor para los arcos no existentes el valor de Short.MAX_VALUE.

Además, si la función a probar incluye más parámetros, los paneles correspondientes a dichos parámetros tendrán la misma forma que en el diálogo del generador de datos original (salvo en el caso en que se trate de una matriz de enteros, pues será igual que el mencionado anteriormente).

En la siguiente imagen se muestra la secuencia para el acceso al diálogo correspondiente a este generador de datos (véase Figura 35).

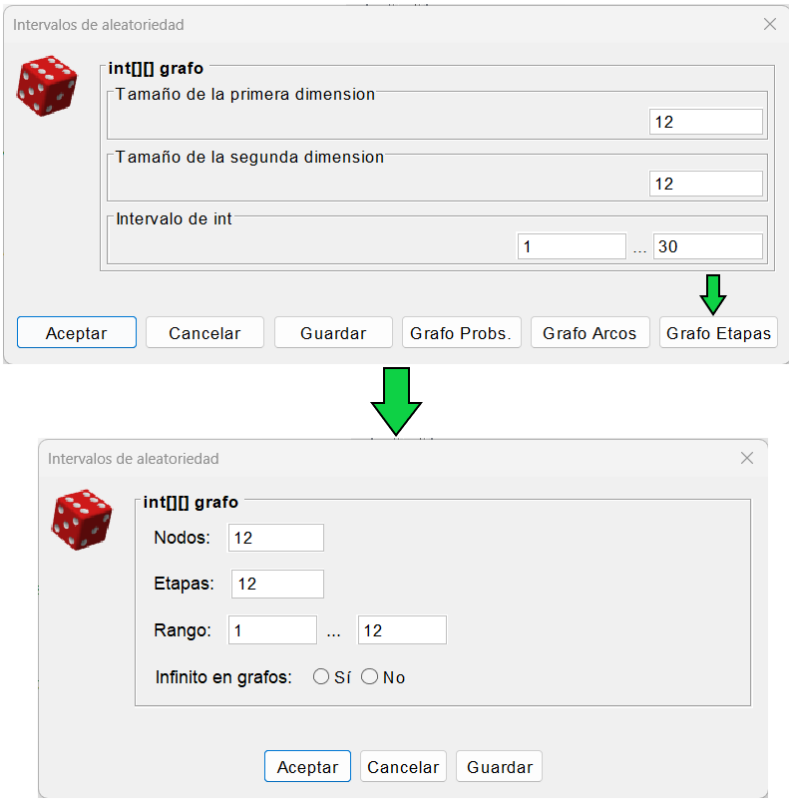


Figura 35: Secuencia de acceso al diálogo del generador de grafos multietapa

Como se puede apreciar, por defecto, el número de nodos es igual a 12 y el número de etapas es igual a 12.

Sin embargo, en el selector que marca si es “Infinito en grafos” o no, no está marcada ninguna opción. Esto es debido a que, como bien se ha comentado con anterioridad, dependiendo de la opción elegida, se muestran unas restricciones u otras.

En la siguiente imagen se muestra la secuencia de posibles restricciones que se muestran dependiendo de la opción pulsada en el selector de “Infinito en grafos” (véase Figura 36).

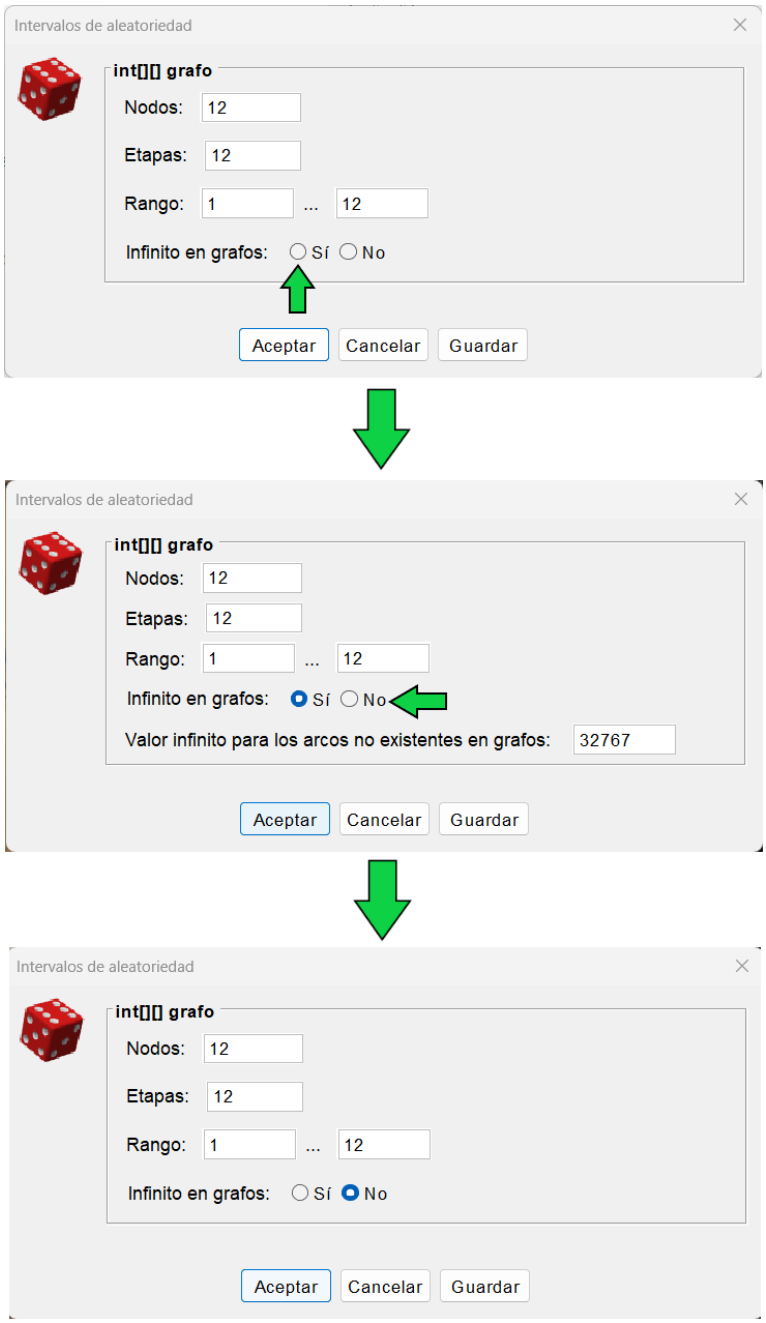


Figura 36: Secuencia de posibles restricciones en el diálogo del generador de grafos multietapa

Para este generador de datos, dependiendo de los valores introducidos o por la falta de opciones seleccionadas, se mostrarán diferentes mensajes de error:

- En el caso de introducir un número de nodos menor que 0, se mostrará el mensaje de error visto en la *Figura 28*.

- Debido a la manera en la que he planteado este generador de datos, el número de nodos (y, por ende, de etapas) debe ser superior a 2, ya que el primer y último nodo se consideran como etapas y no existiría ninguna etapa intermedia.

Por lo tanto, si el número de etapas es menor o igual a 2, o es superior al número de nodos, se mostrará este mensaje de error (véase Figura 37).

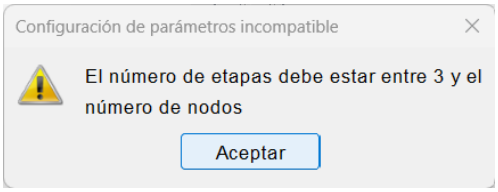


Figura 37: Mensaje de error en caso de introducir un número de etapas inválido (Generador de Grafos Multietapa)

- En el caso de:
    - Dejar el campo del número de nodos vacío.
    - Dejar el campo de número de etapas vacío.
    - No pulsar ninguna opción en el selector de "Valuado".
    - No pulsar ningún botón en el selector de "Infinito en grafos".
    - Pulsar el botón "Sí" en el selector de "Infinito en grafos" y dejar el campo del valor para los arcos inexistentes vacío.

  Se mostrará el mensaje de error visto en la *Figura 35*, indicándonos que debemos completar todos los parámetros del panel.

4.1.8. Almacenamiento de restricciones en los nuevos generadores

Como se puede ver en las figuras vistas a lo largo del apartado, en los nuevos generadores de datos se ha incluido la opción de guardar restricciones, reutilizando y ampliando el funcionamiento de los métodos ya existentes encargados de dicha función.

Al pulsar el botón de "Guardar" en cualquiera de los diálogos de los nuevos generadores de datos (al igual que en el original), se abre un nuevo diálogo en el que se puede seleccionar el directorio y el nombre del fichero a almacenar.

La extensión del fichero es .cnstr, un formato especial para los ficheros que contienen las restricciones.

En el caso de los generadores de permutaciones y el de tareas, como se reutilizan los atributos ya existentes y no requieren el tratamiento de nuevas restricciones, se utiliza el tamaño y el rango de los números a generar, algo ya contemplado en el generador de datos original.

Por lo tanto, el formato del contenido del fichero de extensión .cnstr es igual al generado por el generador de datos original, guardando el tamaño, el rango de números (el mínimo y el

máximo), el orden de los números generados (aleatorio, ascendente o descendente), y si se repiten (o no) los números en el array.

En la siguiente imagen se puede apreciar lo anteriormente comentado, tras haber guardado en el fichero las restricciones mediante el diálogo del generador de permutaciones (véase Figura 38).

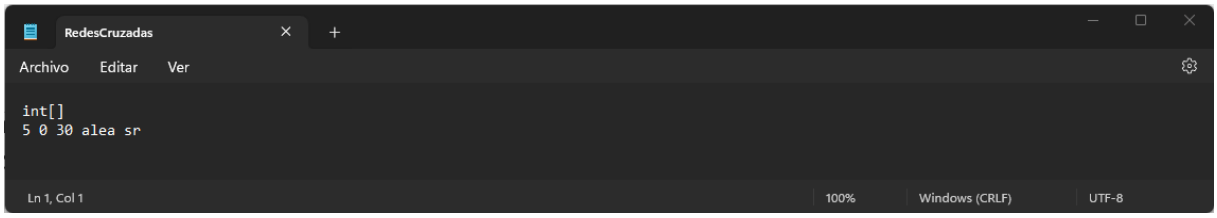


Figura 38: Formato del contenido del fichero de restricciones del generador de permutaciones (aplicable al de tareas)

En la imagen anterior, se ve que el tamaño del array es igual a 5, el valor mínimo es 0, el valor máximo es 30, el orden de los elementos es aleatorio y se admiten repeticiones.

El ejemplo anterior es aplicable al generador de tareas, pues se almacenan los datos de la misma manera, reutilizando lo ya existente para el generador de datos original.

Sin embargo, para el caso de los generadores de grafos es distinto, pues existen más restricciones a manejar para la generación de datos y, por ende, el formato del contenido del fichero de extensión .cnstr correspondiente cambia (en el apartado 5.5 se explica con más detalle el contenido de dicho fichero).

En estos generadores de grafos, además de almacenar los valores ya establecidos (para el caso de utilizar el generador de datos principal), se almacenan nuevos valores, los cuales son: el número de nodos, la probabilidad de arco, el número de arcos, el número de etapas, la dirección del grafo, si el grafo es valuado (o no), el valor mínimo y máximo del rango de valores del grafo (en caso de que sea valuado), si el grafo es reflexivo (o no), si el grafo es infinito en grafos (o no), y por último, el valor de los arcos no existentes en caso de que el grafo sea valuado.

En la siguiente imagen se puede apreciar lo anteriormente comentado, tras haber guardado en el fichero las restricciones mediante el diálogo del generador de grafos por probabilidad de arco (véase Figura 39).

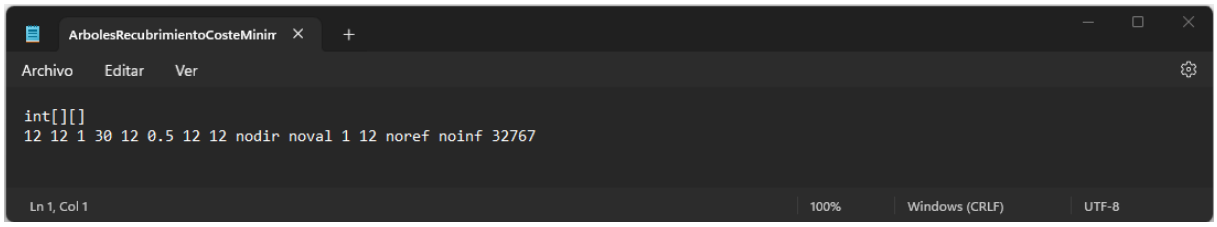


Figura 39: Formato del contenido del fichero de restricciones del generador de grafos por probabilidad de arco (aplicable al resto de generadores de grafos)

En la imagen anterior, se ve que el número de filas de la matriz es 12, el número de columnas es 12, el valor mínimo del rango de valores es 1 y el valor máximo es 30 (estos datos serían para el generador de datos principal).

Para los generadores de grafos, los datos comienzan a partir del quinto elemento, siendo los siguientes: el número de nodos es 12 ; la probabilidad de arco es de 0.5 ; el número de arcos es 12 ; el número de etapas es 12 ; el grafo es no dirigido y no valuado; los valores mínimo y máximo del rango para los arcos existentes en caso de ser valuado son 1 y 12, respectivamente ; el grafo es no reflexivo ; el grafo es no infinito en grafos ; el valor de los arcos no existentes es 32767.

Realmente, a la hora de guardar las restricciones de este caso, las restricciones modificadas han sido la del número de nodos, la de la probabilidad de arco, la de la dirección del grafo, la de si el grafo es valuado (en este caso no lo es) y la de si el grafo es reflexivo (en este caso no lo es).

Por lo tanto, el resto de las restricciones mencionadas toman los valores por defecto, para que, en caso de cargar las restricciones y elegir otras restricciones u otros generadores, también haya opciones por defecto y no existan conflictos.

4.2. Arquitectura

En este apartado de la memoria, se incluye un resumen de la arquitectura del proyecto, de tal forma que se muestra qué clases han sido modificadas o creadas a lo largo del desarrollo del proyecto.

Debido a la gran cantidad de clases que componen el proyecto (49 en total), no resulta factible el hecho de incluir un diagrama de clases con todas las clases del proyecto, pues no quedaría en un formato legible.

Es por ello por lo que, en base a las clases que he modificado y creado a lo largo del desarrollo del proyecto (12 en total), las he recopilado y, gracias a la herramienta de la que dispone IntelliJ IDEA en su versión Ultimate he generado el siguiente diagrama de clases (véase Figura 40).

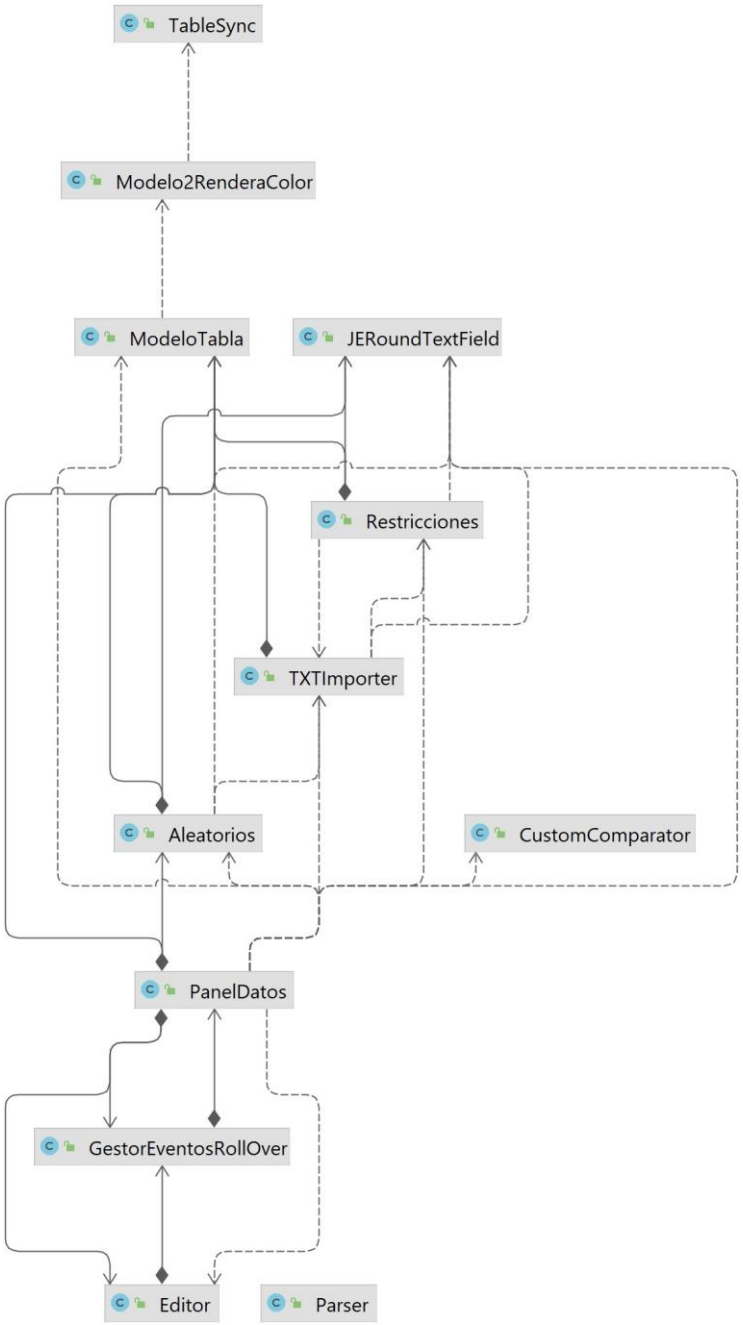


Figura 40: Diagrama de clases simplificado de AlgorEx. Incluye las clases modificadas y creadas durante el desarrollo

Como se puede apreciar en la imagen anterior, las clases que se han modificado son las siguientes:

- **Aleatorios:** en esta **clase** se encuentran buena parte de los **cambios**, pues en ella están los **nuevos generadores de datos** y sus correspondientes **diálogos**.

- **CustomComparator:** en esta **clase** se encuentra el **comparador** que permite la **ordenación** de los elementos de las **tablas de datos y ejecuciones**.

- **Editor:** esta **clase** apenas ha sufrido **cambios**, pero es de vital **importancia** para la **gestión de las acciones** de los **generadores de datos**.

- **GestorEventosRollOver:** esta **clase** apenas ha sido **modificada** (salvo algunos retoques **estéticos** en la estructura del **código**). Sin embargo, es **importante**, debido a que gestiona los **eventos** producidos a la hora de pulsar los **botones** de la **interfaz**.

- **JERoundTextField:** aunque esta **clase** no ha sido **modificada**, es utilizada para buena parte de los **campos de texto** de los **diálogos** de los **generadores de datos**.

- **Modelo2RenderaColor:** esta **clase** se usa para renderizar el **modelo** correspondiente a la **tabla de ejecuciones**. Además, en esta **clase** se establece el **ordenador de filas** (*RowSorter*) para la **tabla de ejecuciones**.

- **ModeloTabla:** esta **clase** es utilizada para las **tablas** utilizadas en **AlgorEx**, pues en ella se gestionan todos los **parámetros** referentes a las **tablas**, incluyendo añadir filas, eliminarlas, moverlas de posición, etc.

- **PanelDatos:** en esta **clase** se gestiona todo lo referente al **modelo** de la **tabla de datos**.

- **Parser:** mediante esta **clase** se pueden realizar transformaciones (*castings*) de cualquier **tipo de dato** a otro **tipo** de los **tipos básicos** de **Java**. Por ejemplo, para convertir un **String** a **int**, se puede utilizar la función **StringAEntero**. Como se verá más adelante, esta **clase** se ha modificado para la **obtención** de **caracteres Unicode**.

- **Restricciones:** de igual manera que la clase **Aleatorios**, en esta **clase** se encuentran buena parte de los **cambios**, pues en ella están los **nuevos generadores de datos** y sus correspondientes **diálogos**. Sin embargo, se utiliza para el caso en el que se carguen **restricciones** a la hora de generar **juegos de datos**.

- **TableSync:** esta **clase** no estaba en la **versión anterior** de **AlgorEx** y ha sido creada, como más tarde se explicará en profundidad, para **sincronizar la ordenación** entre la **tabla de datos** y la **tabla de ejecuciones**.

- **TXTImporter:** en esta **clase** se concentran los **métodos** que permiten la **carga** y **almacenamiento** de los **ficheros de texto**. Ha sido modificada para el **almacenamiento** y **carga** de las **restricciones** para los **nuevos generadores de datos**.

5. Implementación

En este apartado de la memoria se incluye la implementación de los cambios desarrollados en el proyecto para la inclusión de las mejoras y los nuevos generadores de datos en el sistema AlgorEx. Además, se incluirán enlaces referenciando a los códigos mencionados, los cuales están disponibles en el *Anexo I*.

Así pues, en este apartado se incluye: el funcionamiento de AlgorEx en diferentes JDK, la reordenación de las tablas de datos y ejecuciones, los cambios en la generación aleatoria de datos, y los nuevos generadores de datos (incluyendo el acceso a los mismos y el almacenamiento de restricciones de estos).

5.1. Funcionamiento de AlgorEx en diferentes JDK

Como el primer objetivo era comprobar que AlgorEx funciona con diferentes versiones de JDK (Java Development Kit) desde la versión 6 en adelante, la solución que encontré fue cambiar, dentro de la configuración del proyecto, el SDK (*Software Development Kit*) al JDK 19.0.2, con compatibilidad de lenguaje a la versión 8 de Java, que es la que me permitía solventar dicho problema.

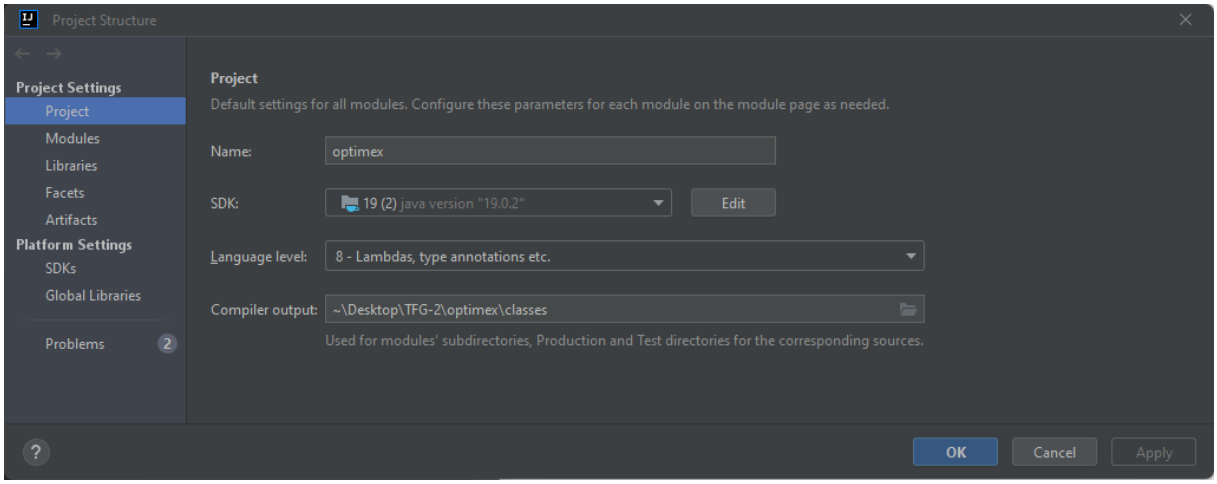


Figura 41: Configuración de la estructura del proyecto

5.2. Ordenación de las tablas de datos y ejecuciones

Respecto a los objetivos referentes a la ordenación de las tablas de datos y de ejecuciones, hice lo siguiente:

- Para que la tabla de ejecuciones se pudiese ordenar por cualquier columna, lo que hice fue llamar al método *setCustomRowSorter* (más tarde será explicado) en el método *getTableCellRendererComponent* de la clase *Modelo2RenderaColor* (en la que se renderiza el modelo de la tabla de ejecuciones), justo cuando se comprueba si el *RowSorter* del modelo de la tabla de ejecuciones es igual a *null*.

- Para la cuestión de la sincronización del orden de los elementos de las tablas de datos y de ejecuciones, el proceso fue el siguiente:

    - Debido a que debe sincronizarse la ordenación de las tablas de datos y de ejecuciones, se hacía necesario poder gestionar los eventos de ordenación de filas de ambos modelos de tabla.

    - Por ello, creé la clase *TableSync*, la cual implementa los métodos de la interfaz *RowSorterListener*, mediante la que se puede "escuchar" los eventos de ordenación de filas. Esta clase tiene los siguientes atributos:

        - **JTable table:** este atributo se corresponde con la tabla de la que se va a obtener el orden de los elementos.

        - **JTable otherTable:** este atributo se corresponde con la tabla sobre la que se va a aplicar el orden de los elementos.

        - **int columnIndex:** este atributo es el número de columna del que se toman referencias del orden de los elementos.

        - **int count:** este atributo se utiliza para contar el número de ordenaciones que se hacen (se inicializa a 0).

            - Debido a que cada vez que se ordena una tabla por cualquiera de sus columnas, entra a esta clase dos veces (una para obtener el orden previo y otra para obtener el orden posterior), se utiliza para quedarnos con solo la segunda ordenación (como más tarde se verá).

    - Sin embargo, había otro problema: pese a que se podían obtener los eventos de ordenación de cualquiera de las tablas, no era posible aplicar dicha ordenación a la otra tabla.

    - Tras mucho investigar, vi que en interfaces de modelos de tabla como la clase *DefaultTableModel* (de Java) existía un método que permitía mover las filas de un índice a otro de la tabla, llamado *moveRow*.

    - Como la clase *ModeloTabla* hereda de la clase *AbstractTableModel* y no dispone de dicho método, incluí un método llamado *moverFila* que replica el funcionamiento del método *moveRow*.

    - Este método tiene como parámetros dos enteros, que indican la posición desde la que se mueve la fila y la posición hacia la que se mueve, respectivamente.

Pulsa aquí para acceder al código

- En este método, primero se comprueba que *'desde'* y *'hasta'* son iguales, en cuyo caso no sería necesario hacer nada.

  A continuación, obtenemos la fila que se elimina de los datos de la tabla y se añade de nuevo en la posición *'hasta – (desde < hasta ? 1 : 0)'* (teniendo en cuenta si *'desde'* es menor o mayor que *'hasta'*).

  Tras ello, se notifica a los *listeners* del modelo de tabla la eliminación y la inserción de las filas (mediante *'fireTableRowsDeleted'* y *'fireTableRowsInserted'*), además de notificar la actualización de las filas que se han movido (mediante *'fireTableRowsUpdated'*).

- Tenemos el siguiente algoritmo en el método *sorterChanged* de la clase *TableSync* (para detectar cambios de orden de filas en las tablas):

  Pulsa aquí para acceder al código

  En este caso, por cada alteración del orden de las filas de las tablas, se aumenta el parámetro *count* y, en caso de que sea par, se sincroniza el orden de ambos modelos de tabla (por lo explicado anteriormente).

  - Se crea una lista de enteros en la que, al recorrer las filas de la tabla, se guardan los valores de la columna de *columnIndex*.

  - Se obtiene el modelo de la tabla a la que se transferirá el orden de las filas y se recorre la lista que contiene los valores ordenados de la columna indicada.

    Dentro de ese bucle for, se recorren las filas de la otra tabla y, si coincide el elemento de la lista con el valor de la columna indicada en la fila de la otra tabla, se moverá la fila de la posición actual a la posición del elemento en la lista.

- Para que este método se aplique en cada ordenación que se haga tanto en la tabla de datos como en la tabla de ejecuciones, lo que hice fue asignarle el *listener* a cada tabla tras establecer el *RowSorter* a la tabla de ejecuciones en el método *getTableCellRendererComponent* de la clase *Modelo2RenderaColor*.

  Pulsa aquí para acceder al código

  Esto se hace llamando al método *addRowSorterListener* del que dispone el *RowSorter* correspondiente a cada una de las tablas.

- Para que las tablas puedan ser ordenadas en cualquier caso, también se han incluido cambios en el método *setCustomRowSorter* de la clase *PanelDatos*, para que sean aceptados también los números enteros (incluyendo los *arrays*) y los números reales.

Pulsa aquí para acceder al código

Se ha incluido un parámetro de tipo entero mediante el cual se identifica la tabla a la que se lo estamos aplicando.

Si el número de tabla es igual a 1, se aplica sobre la tabla de datos y, por ende, solo nos interesa aplicar el *CustomComparator* en los casos en que los datos de la columna sean números.

En el resto de las tablas se aplicará el comparador para todas las columnas (realmente se aplica sólo sobre la tabla de ejecuciones, que por lo general ofrece resultados numéricos).

- De igual manera, también se han aplicado cambios en el método compare de la clase CustomComparator, para que se puedan comparar tanto arrays (de una o dos dimensiones) como números (ya sean enteros o reales):

Pulsa aquí para acceder al código

5.3. Cambios en la generación aleatoria de datos

En referencia a los objetivos correspondientes a la generación aleatoria de datos, hice lo siguiente:

- Uno de los objetivos era que, para generar datos de tipo *char*, el generador de datos admitiera los formatos válidos en Java, incluso caracteres Unicode.

  Para ello, debido al cambio de formato solicitado, lo que hice fue cambiar los atributos correspondientes al valor mínimo (*charinf*) y al valor máximo (*charsup*) del rango de valores de los caracteres en el diálogo del generador de datos (tanto en la clase *Aleatorios* como en la clase *Restricciones*).

  Pulsa aquí para acceder al código

  Los valores anteriores eran 0 y z, sin las comillas simples. Una vez adaptados dichos atributos, el siguiente paso fue cambiar el funcionamiento del método *StringACaracter* de la clase *Parser* para el nuevo formato de los datos.

  Pulsa aquí para acceder al código

  En primer lugar, se eliminan los posibles espacios en blanco que pueda tener la cadena. Tras ello, tenemos diferentes casos:

  - Si dicha cadena está vacía, se devolverá un espacio.

  - Si la longitud de la cadena es mayor a 1, se obtendrá la cadena sin las comillas simples.

- Se obtiene el *char* correspondiente a la cadena resultante y:
    - Si la cadena contiene la barra invertida, se valoran los siguientes casos:
        - Si la longitud de la cadena es menor que 2, directamente se devuelve dicha cadena.
        - En caso contrario, se valoran los diferentes casos de las secuencias de escape de Java:
            - **\' :** comilla simple (apóstrofe).
            - **\":** comilla doble.
            - **\\:** barra invertida.
            - **\n:** nueva línea.
            - **\r:** retorno de carro.
            - **\t:** tabulación horizontal.
            - **\b:** retroceso.
            - **\f:** avance de página.
            - **\uXXXX:** valor Unicode de 4 dígitos hexadecimales (por ejemplo, \u0068 representa el carácter Unicode 'h'). En este caso particular, se extrae la subcadena correspondiente a dichos dígitos y se realiza la conversión a entero, para obtener el carácter.
    - En caso contrario, se devuelve directamente la cadena obtenida.

- Otro de los objetivos era solucionar el problema de los juegos de datos repetidos, pues en la versión anterior de AlgorEx se generaban menos juegos de datos que los indicados. Para ello, hice los siguientes cambios:
    - Este problema se producía en el método *anhadeFilaAleatoria* de la clase *ModeloTabla*, pues en el caso en el que la fila existiese ya, no trataba de generar de nuevo un juego de datos aleatorio, por lo que siempre se generaban menos juegos de datos que los indicados.

Pulsa aquí para acceder al código

- Debido a los diferentes generadores de datos que se han desarrollado (los cuales se explicarán más adelante), ha sido necesario modificar este método, incluyendo dos nuevos parámetros de tipo entero: *opcion* y *version*.

  Mediante el parámetro *opcion* se gestiona el generador de datos al que llamar en caso de que la fila generada exista ya en la tabla. Por otro lado, el parámetro *version* nos permite gestionar a qué versión del generador de datos llamar en caso de que la fila generada exista ya en la tabla (la de la clase *Aleatorios* o la de la clase *Restricciones*).

  Así pues, la nueva versión del método *anhadeFilaAleatoria* queda de la siguiente manera, solucionando el problema antes mencionado:

  Pulsa aquí para acceder al código

  Como se puede apreciar, todos los métodos de los generadores de datos (incluyendo el del generador de datos principal) se han desarrollado como métodos públicos y estáticos, para que fuesen accesibles sin necesidad de crear una instancia de la clase correspondiente, ya fuese la clase *Aleatorios* o la clase *Restricciones*.

5.4. Nuevos generadores de datos

Tras incluir los cambios mencionados con respecto al botón de guardar restricciones, el último de los elementos incluidos en la especificación era el de integrar nuevos generadores de datos.

Dado que estos nuevos generadores de datos deben ser para funciones con todo tipo de parámetros (cumpliéndose las condiciones para los mismos), se hizo necesario modularizar el proceso de creación de diálogos y de generación de juegos de datos que corresponden al generador de datos principal.

Por ello, del método *actualizarTopes* de la clase *Aleatorios* (y de su homólogo en la clase *Restricciones*, el método *actualizaCotas*) se han extraído diferentes partes para generar los siguientes métodos:

- **inicializadorTopesConTiposIguales(String[] tipos):** en este método se inicializan los valores de los topes de las restricciones en caso de que los tipos de los parámetros de la función a probar sean iguales a los de la función probada anteriormente. Sirve para cuando se generan tandas separadas de juegos de datos sobre la misma función, agilizando el proceso.

- **inicializarTopes(String[] tipos):** en este método se inicializan los valores de los topes de las restricciones con los valores de una serie de atributos de la clase (*intinf, intsup,* etc.) para la primera vez en la que se prueba una función.

- **generarPanelParametro(String[] tipos, int i):** este método es utilizado para generar el panel de un parámetro concreto de la función que se está probando, para que este

sea agregado al diálogo correspondiente. Se utiliza para todos aquellos parámetros que no están relacionados con los nuevos generadores de datos.

- **establecerTopesUsuario(String[] tipos, int i):** este método sirve para almacenar los topes y restricciones establecidos por el usuario en el diálogo del generador de datos correspondiente.

- **generarDatosAleatoriosColumna(String[] tipos, String[] datos, int i):** este método sirve para generar los datos de una columna en concreto de una fila de la tabla de datos. Se utiliza para aquellos parámetros que no relacionados con los nuevos generadores de datos.

Debido a que la implementación de estos métodos ya estaba incluida en el código de la versión anterior de AlgorEx y no aportan nada nuevo, no es necesario incluir su implementación.

5.4.1. Implementación del generador de tareas

Para el correcto funcionamiento de este generador de datos, he dividido el código en cuatro métodos:

- **ventanaTareas(String[] tipos):** en este método se genera el diálogo correspondiente a este generador de datos, para que el usuario pueda insertar las restricciones de los parámetros de la función a probar.

Pulsa aquí para acceder al código

- **generarPanelTareas(String[] tipos, int posicionPrimerArray, int posicionSegundoArray):** se trata de un método privado que se encarga de generar el panel correspondiente a los *arrays* de las tareas en la ventana de tareas (utilizado en el método anterior).

Pulsa aquí para acceder al código

- **generarFilasTareas():** este método público se encarga de generar los datos para las filas de la tabla de datos en el caso del generador de tareas.

Pulsa aquí para acceder al código

El funcionamiento del método es el siguiente:

  - Se crean dos *arrays* de *Strings*: *tipos* y *datos*, cuya longitud es igual al número de columnas de la tabla de datos.

  - Se recorre cada columna de la tabla de datos y se agrega el tipo de dato de cada columna al *array* de tipos.

- Se ubica la posición de los *arrays* de enteros en el array de tipos utilizando *streams* y se almacena en la lista de enteros llamada *posicionesArraysEnteros*.

- Se obtienen las posiciones del primer y segundo *array* de enteros en el array de tipos utilizando la lista *posicionesArraysEnteros*.

- Se obtiene el valor de los *arrays* correspondientes a los de las tareas y se almacena en un entero llamado *topeTareas*.

- Se crean dos *arrays* de enteros, *cs* y *fs*, con un tamaño igual a *topeTareas*.

- Se asignan los valores a los *arrays cs* y *fs* utilizando el método *asignarValoresTareas*, pasando la posición del primer array, el tope de tareas, y los arrays *cs* y *fs*.

- Se asignan los arrays *cs* y *fs* a las posiciones correspondientes en el array de datos.

- Se generan datos aleatorios para el resto de los parámetros que no son *arrays* de enteros utilizando el método *generarDatosAleatoriosColumna*.

- Se crea un objeto de la clase *Fila* con los datos generados y se agrega a la tabla de datos utilizando el método *anhadeFilaAleatoria* de la clase *ModeloTabla*.

- **asignarValoresTareas(int posicionPrimerArray, int topeTareas, int[] cs, int[] fs):** este método privado se encarga de asignar los valores a los *arrays* de enteros correspondientes a los instantes de comienzo (*cs*) y los instantes de final (*fs*), siendo el valor mayor para el *array cs* y el menor para el *array fs*.

  Pulsa aquí para acceder al código

  El funcionamiento del método es el siguiente:

  - Se inicia un bucle for desde 0 hasta el tope de los *arrays* de tareas.

  - Se generan dos enteros aleatorios (*entero1* y *entero2*) en los rangos correspondientes a cada *array* de las tareas.

  - Mientras sean iguales, se seguirán generando hasta que sean distintos.

  - Una vez pasada dicha comprobación, el menor de los dos valores irá para el *array cs* y el mayor para el *array fs*.

5.4.2. Implementación del generador de permutaciones

Para el correcto funcionamiento de este generador de datos, he dividido el código en cinco métodos:

- **ventanaPermutaciones(String[] tipos):** en este método se genera el diálogo correspondiente a este generador de datos, para que el usuario pueda insertar las restricciones de los parámetros de la función a probar.

Pulsa aquí para acceder al código

- **generarPanelPermutaciones(String[] tipos, int i):** se trata de un método privado el cual se encarga de generar el panel correspondiente al *array* sobre el que se generarán las permutaciones para la ventana de permutaciones (utilizado en el método anterior).

Pulsa aquí para acceder al código

- **generarFilasPerms():** este método público se encarga de generar los datos para las filas de la tabla de datos en el caso del generador de permutaciones.

Pulsa aquí para acceder al código

El funcionamiento del método es el siguiente:

  - Se crean dos *arrays* de String: *tipos* y *datos*, cuya longitud es igual al número de columnas de la tabla de datos.

  - Se recorre cada columna de la tabla de datos y se agrega el tipo de dato de cada columna al *array* de tipos.

  - De nuevo, se recorre cada columna de la tabla de datos y se realiza lo siguiente:

    - Si el tipo de la columna es un *array* de enteros (int[]), se hace lo siguiente:

      - Se obtiene el tope del *array* a generar.

      - Con dicho tope se obtienen las permutaciones llamando al método *obtenerPermutaciones*.

      - El *array* resultante se almacena en la posición *i* del *array* de datos.

    - En caso contrario, se generan datos aleatorios para el resto de los parámetros utilizando el método *generarDatosAleatoriosColumna*.

  - Se crea un objeto de la clase *Fila* con los datos generados y se agrega a la tabla de datos utilizando el método *anhadeFilaAleatoria* de la clase *ModeloTabla*.

- **obtenerPermutaciones(int tope):** este método privado se utiliza para generar un *array* de permutaciones en base a un tope dado.

Pulsa aquí para acceder al código

El funcionamiento del método es el siguiente:

- Se crea una lista de enteros llamada *listaPerms* para guardar las permutaciones.
- Se realiza un bucle for desde 0 hasta el valor del tope y se realiza lo siguiente:
  - Se genera un número entero aleatorio en el rango de 0 a tope – 1 (ambos inclusive).
  - Mientras que el valor generado esté en la lista (*listaPerms*), se genera un nuevo entero aleatorio en el rango especificado.
  - Tras haber obtenido un entero aleatorio único, se agrega a la lista.
- Se crea un nuevo *array* de enteros llamado *arrayPerms*, con un tamaño igual al tope.
- Se recorre la lista (*listaPerms*) y se asigna cada permutación a la posición correspondiente en el *array* (*arrayPerms*).
- Se devuelve el *array* de permutaciones (*arrayPerms*).

- **factorial(int n):** mediante este método privado se obtiene el factorial de un número dado utilizando recursión.

Pulsa aquí para acceder al código

El funcionamiento del método es el siguiente:

- Si el número *n* es igual a 0, se devuelve un *BigInteger* con el valor de 1, ya que el factorial de 0 es 1.
- En caso contrario, se realiza lo siguiente:
  - Se crea un *BigInteger* con el valor de *n*.
  - Se multiplica dicho *BigInteger* por el resultado de la llamada al método *factorial* con el número n-1 (mediante llamadas recursivas).
  - El resultado de dicha multiplicación se devuelve como el resultado del método.

En este caso, se ha utilizado el tipo *BigInteger* en lugar de *int* para aquellos casos en los que el resultado del factorial fuese demasiado grande como para almacenarlo en un entero normal.

5.4.3. Cambios para la inclusión de los generadores de grafos

Para los generadores de datos referentes a los grafos, ha sido necesario crear diferentes atributos tanto para la clase *Aleatorios* como para la clase *Restricciones*.

- Esta serie de atributos hacen referencia a los *textField*, los grupos de botones, algunos botones particulares, los booleanos que indican propiedades de los grafos (dirección del grafo, valuación, reflexividad, etc.) y los títulos referentes a las restricciones que se manejan en los diálogos de los diferentes generadores de datos.

- Estos atributos se inicializan en el método *actualizarTopes* de la clase *Aleatorios* (método *actualizaCotas* en la clase *Restricciones*), pues cuando se está probando una función en la que los tipos no son iguales a los almacenados en el *recordador*, y si dentro de los parámetros hay una matriz de enteros, se llama al método *inicializarDatosGrafos*.

Pulsa aquí para acceder al código

5.4.4. Implementación del generador de grafos por probabilidad de arco

Para el correcto funcionamiento de este generador de datos, he dividido el código en cuatro métodos:

- **ventanaGrafoProbs(String[] tipos):** en este método se genera el diálogo correspondiente a este generador de datos, para que el usuario pueda insertar las diferentes restricciones de los parámetros de la función a probar.

Pulsa aquí para acceder al código

- **generarPanelGrafoProbs(String[] tipos, int i):** se trata de un método privado el cual se encarga de generar el panel correspondiente a la matriz de enteros sobre la que se generará el grafo para la ventana de grafo por probabilidades. Este método es utilizado en el método *ventanaGrafoProbs*.

Pulsa aquí para acceder al código

- **generarFilasGrafoProbs():** este método público se encarga de generar los datos para las filas de la tabla de datos en el caso del generador de grafos por probabilidad de arco.

Pulsa aquí para acceder al código

  El funcionamiento del método es el siguiente:

  - Se crean dos *arrays* de String: *tipos* y *datos*, cuya longitud es igual al número de columnas de la tabla de datos.

- Se recorre cada columna de la tabla de datos y se agrega el tipo de dato de cada columna al *array* de tipos.

- De nuevo, se recorre cada columna de la tabla de datos y se realiza lo siguiente:

    - Si el tipo de la columna es una matriz de enteros (int[][]), se generan datos aleatorios en base a las restricciones establecidas por el usuario llamando al método *generarDatosColumnaGrafoProbs*.

    - En caso contrario, se generan datos aleatorios para el resto de los parámetros utilizando el método *generarDatosAleatoriosColumna*.

- Se crea un objeto de la clase *Fila* con los datos generados y se agrega a la tabla de datos utilizando el método *anhadeFilaAleatoria* de la clase *ModeloTabla*.

- **generarDatosColumnaGrafoProbs(String[] datos, int i):** se trata de un método privado que se encarga de generar los datos aleatorios de una columna en concreto de una fila de la tabla de datos. En este caso, es para un grafo generado a partir de la probabilidad de arco.

Pulsa aquí para acceder al código

De forma resumida, el funcionamiento de este código es el siguiente:

- Se obtiene el número de nodos (*numNodos*) y la probabilidad de arco (*probabilidad*) a partir de los *textfield* correspondientes.

- Se crea una matriz de enteros (*grafo*) correspondiente con la matriz de adyacencia del grafo, que tiene por dimensiones *numNodos* por *numNodos*.

- Se almacenan en sendas variables booleanas (*esGrafoValuado* y *esInfinitoEnGrafo*) para saber si el grafo es valuado y si es infinito en grafos.

- Si el grafo es valuado y es infinito en grafos, se inicializan los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor introducido por el usuario en el campo correspondiente.

- Por otro lado, si el grafo es valuado, pero no es infinito en grafos, se inicializan los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor Short.MAX_VALUE (32767).

- Se obtiene la dirección del grafo (en el array de booleanos llamado *direccionGrafo*) y si el grafo es reflexivo (en el booleano *esGrafoReflexivo*).

- Dependiendo de la dirección del grafo (no dirigido, dirigido acíclico o dirigido acíclico), se asignan los valores (arcos) correspondientes en la matriz *grafo*.

- Si el grafo es no dirigido, tenemos los siguientes casos:

  - Si es no valuado y reflexivo, se asigna 1 a los elementos de la matriz *grafo* en los que la probabilidad generada aleatoriamente sea menor que la establecida por el usuario.

  - En caso contrario, tenemos dos casos para el caso en que la probabilidad generada aleatoriamente sea menor que la establecida por el usuario: cuando el grafo sea valuado, se asigna un valor al arco en el rango establecido por el usuario; y cuando el grafo sea no valuado, se le asigna 1 al arco.

  - Dado que en el caso contrario no es reflexivo, los arcos en la diagonal de la matriz (es decir, los arcos a sí mismo) no se insertan.

  - Como se trata de un grafo no dirigido, se recorre la mitad triangular superior de la matriz y los arcos se establecen recíprocamente unos con otros. Es decir, se inserta el valor en el arco [i][j] y en el [j][i].

- Si el grafo es dirigido cíclico, se recorren las posiciones de la matriz y tenemos los siguientes casos en caso de que la probabilidad generada aleatoriamente sea menor que la establecida por el usuario:

  - Si el número de fila es distinto al de columna, se establece un valor al arco en el rango establecido por el usuario si el grafo es valuado, y en caso contrario, se asigna un 1 al arco correspondiente.

  - En caso contrario, se comprueba si el grafo es no valuado y reflexivo, y si se cumple la condición, se asigna un 1 al arco correspondiente.

- Si el grafo es dirigido acíclico, tenemos los siguientes casos:

  - Si es no valuado y reflexivo, se recorren las posiciones de la matriz y se asigna 1 a los elementos de la matriz *grafo* en los que la probabilidad generada aleatoriamente sea menor que la establecida por el usuario y en el caso en que el número de fila sea menor o igual que el número de columna.

    Esto último se debe a que no se permiten ciclos (salvo consigo mismo al ser reflexivo) y se deben establecer arcos en la mitad triangular superior de la matriz.

  - En caso contrario, se recorre la mitad triangular superior de la matriz *grafo* y tenemos dos casos para el caso en que la

probabilidad generada aleatoriamente sea menor que la establecida por el usuario: cuando el grafo sea valuado, se asigna un valor al arco en el rango establecido por el usuario; y cuando el grafo sea no valuado, se le asigna 1 al arco.

- ❖ Dado que en el caso contrario no es reflexivo, los arcos en la diagonal de la matriz (es decir, los arcos a sí mismo) no se insertan.

- Se transforma la matriz resultante a un *String* y se almacena en la posición *i* del *array* de datos.

5.4.5. Implementación del generador de grafos por el número de arcos

Para el correcto funcionamiento de este generador de datos, he dividido el código en cinco métodos:

- **ventanaGrafoArcos(String[] tipos):** en este método se genera el diálogo correspondiente a este generador de datos, para que el usuario pueda insertar las diferentes restricciones de los parámetros de la función a probar.

Pulsa aquí para acceder al código

- **generarPanelGrafoArcos(String[] tipos, int i):** se trata de un método privado el cual se encarga de generar el panel correspondiente a la matriz de enteros sobre la que se generará el grafo para la ventana de grafo por número de arcos. Este método es utilizado en el método *ventanaGrafoArcos*.

Pulsa aquí para acceder al código

- **generarFilasGrafoArcos():** este método público se encarga de generar los datos para las filas de la tabla de datos en el caso del generador de grafos por el número de arcos.

Pulsa aquí para acceder al código

De forma resumida, el funcionamiento de este código es el siguiente:

- Se crean dos *arrays* de String: *tipos* y *datos*, cuya longitud es igual al número de columnas de la tabla de datos.

- Se recorre cada columna de la tabla de datos y se agrega el tipo de dato de cada columna al *array* de tipos.

- De nuevo, se recorre cada columna de la tabla de datos y se realiza lo siguiente:

        - Si el tipo de la columna es una matriz de enteros (int[][]), se generan datos aleatorios en base a las restricciones establecidas por el usuario llamando al método *generarDatosColumnaGrafoArcos*.

        - En caso contrario, se generan datos aleatorios para el resto de los parámetros utilizando el método *generarDatosAleatoriosColumna*.

    - Se crea un objeto de la clase Fila con los datos generados y se agrega a la tabla de datos utilizando el método *anhadeFilaAleatoria* de la clase *ModeloTabla*.

- **generarDatosColumnaGrafoArcos(String[] datos, int i):** se trata de un método privado que se encarga de generar los datos aleatorios de una columna en concreto de una fila de la tabla de datos. En este caso, es para un grafo generado a partir del número de arcos.

Pulsa aquí para acceder al código

De forma resumida, el funcionamiento de este código es el siguiente:

- Se obtiene el número de nodos (*numNodos*) y el número de arcos (*numArcos*) a partir de los textfield correspondientes.

- Se obtiene la dirección del grafo (en el array de booleanos llamado *direccionGrafo*), si el grafo es valuado (en el booleano *esGrafoValuado*) y si el grafo es reflexivo (en el booleano *esGrafoReflexivo*).

- Se almacena en el *array* de enteros *arcos* el resultado de la llamada al método *generarArcosGrafo*, mediante el cual se obtienen los arcos a asignar al grafo.

- Se crea una matriz de enteros (*grafo*) correspondiente con la matriz de adyacencia del grafo, la cual tiene por dimensiones *numNodos* por *numNodos*.

- Se obtiene si el grafo es infinito en grafos (en el booleano llamado *esInfinitoEnGrafo*).

- Si el grafo es valuado y es infinito en grafos, se inicializan todos los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor introducido por el usuario en el campo.

- Por otro lado, si el grafo es valuado, pero no es infinito en grafos, se inicializan los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor Short.MAX_VALUE (32767).

- Dependiendo de la dirección del grafo (no dirigido, dirigido acíclico o dirigido acíclico), se asignan los valores (arcos) correspondientes en la matriz *grafo*.

- Si es no dirigido, se recorre el array de arcos, obteniendo su posición en la matriz *grafo* (fila y columna). En este caso, tenemos dos casos:

  - ❖ Si es un grafo valuado, se asigna un valor al arco en el rango establecido por el usuario.

  - ❖ En caso contrario, se asigna un 1 al arco correspondiente.

  - ❖ Como se trata de un grafo no dirigido, los arcos se establecen recíprocamente unos con otros. Es decir, se inserta el valor en el arco [fila][columna] y en el arco [columna][fila].

- Tanto si es dirigido cíclico como dirigido acíclico, como eso se gestiona en el método *generarArcosGrafo*, se recorre el *array* de arcos, obteniendo su posición en la matriz *grafo* (fila y columna). Tenemos dos casos:

  - ❖ Si es un grafo valuado, se asigna un valor al arco en el rango establecido por el usuario.

  - ❖ En caso contrario, se asigna un 1 al arco correspondiente.

- Se transforma la matriz resultante a un *String* y se almacena en la posición *i* del *array* de datos.

- **generarArcosGrafo(int numNodos, int numArcos, boolean[] direccionGrafo, boolean esGrafoReflexivo):** se trata de un método privado que sirve para generar un *array* con los arcos para un grafo.

Pulsa aquí para acceder al código

De forma resumida, el funcionamiento de este código es el siguiente:

- Se crea una lista de enteros (*listaArcos*) para almacenar los arcos generados.

- Se realiza un bucle for que va desde 0 hasta numArcos (este último exclusive).

  - Dentro del bucle se genera un entero aleatorio (llamado *arcoAleatorio*) entre 0 y $numNodos^2$ - 1 en un bucle do-while.

  - Tras generar el arco, se hacen diferentes comprobaciones para los diferentes tipos de grafo:

    - ❖ Si el grafo es no reflexivo, se obtienen la fila y columna que ocuparían en la matriz y, si son iguales, se cambia el valor de *arcoAleatorio* a -1, para indicar que debe generarse otro arco.

- ❖ Por otro lado, si el grafo es no dirigido o dirigido acíclico, como los arcos se deben generar sobre la mitad triangular superior de la matriz, se obtienen la fila y la columna que ocuparían y, si la fila es mayor que la columna, se cambia el valor de *arcoAleatorio* a -1, para indicar que debe generarse otro arco.

  - Este bucle do-while se repite mientras que el arco esté contenido en la lista de arcos o el arco sea igual a -1 (como se ha mencionado anteriormente).

  - Una vez que se genera un arco válido, se agrega a la lista de arcos.

- Tras completar el bucle for, se crea un *array* llamado *arcos*, cuyo tamaño es igual al número de arcos indicado por el usuario.

- Mediante otro bucle for se itera sobre los arcos contenidos en la lista de arcos y se asigna cada uno al *array arcos*.

- Por último, se devuelve el *array arcos*.

5.4.6. Implementación del generador de grafos multietapa

Para el correcto funcionamiento de este generador de datos, he dividido el código en cinco métodos:

- **ventanaGrafoEtapas(String[] tipos):** en este método se genera el diálogo correspondiente a este generador de datos, para que el usuario pueda insertar las diferentes restricciones de los parámetros de la función a probar.

  Pulsa aquí para acceder al código

- **generarPanelGrafoEtapas(String[] tipos, int i):** se trata de un método privado el cual se encarga de generar el panel correspondiente a la matriz de enteros sobre la que se generará el grafo para la ventana de grafo por número de etapas. Este método es utilizado en el método *ventanaGrafoEtapas*.

  Pulsa aquí para acceder al código

- **generarFilasGrafoEtapas():** este método público se encarga de generar los datos para las filas de la tabla de datos en el caso del generador de grafos por el número de etapas.

  Pulsa aquí para acceder al código

  De forma resumida, el funcionamiento de este código es el siguiente:

- Se crean dos *arrays* de String: *tipos* y *datos*, cuya longitud es igual al número de columnas de la tabla de datos.

- Se recorre cada columna de la tabla de datos y se agrega el tipo de dato de cada columna al array de tipos.

- De nuevo, se recorre cada columna de la tabla de datos y se realiza lo siguiente:

    - Si el tipo de la columna es una matriz de enteros (int[][]), se generan datos aleatorios en base a las restricciones establecidas por el usuario llamando al método *generarDatosColumnaGrafoEtapas*.

    - En caso contrario, se generan datos aleatorios para el resto de los parámetros utilizando el método generarDatosAleatoriosColumna.

- Se crea un objeto de la clase *Fila* con los datos generados y se agrega a la tabla de datos utilizando el método *anhadeFilaAleatoria* de la clase *ModeloTabla*.

- **generarDatosColumnaGrafoEtapas(String[] datos, int i):** se trata de un método privado que se encarga de generar los datos aleatorios de una columna en concreto de una fila de la tabla de datos. En este caso, es para un grafo generado a partir del número de etapas.

  Pulsa aquí para acceder al código

  De forma resumida, el funcionamiento de este código es el siguiente:

  - Se obtiene el número de nodos (*numNodos*) y el número de etapas (*numEtapas*) a partir de los *textfield* correspondientes.

  - Se crea una matriz de enteros (*grafo*) correspondiente con la matriz de adyacencia del grafo, la cual tiene por dimensiones *numNodos* por *numNodos*.

  - Se obtiene si el grafo es infinito en grafos (en el booleano llamado *esInfinitoEnGrafo*).

  - Como el grafo es valuado, tenemos dos casos:

    - Si el grafo es infinito en grafos, se inicializan los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor introducido por el usuario.

    - En caso contrario, se inicializan los elementos de la matriz *grafo* (salvo los de la diagonal) con el valor Short.MAX_VALUE (32767).

- Se obtiene un mapa (*mapaEtapasNodos*), en el que almacenamos las etapas como clave y una lista de enteros que contiene los nodos como valor, todo ello llamando al método *obtenerNodosPorEtapa*.

- Se recorre, mediante un bucle for, las etapas almacenadas en el mapa *mapaEtapasNodos*:

  - Si la etapa es la primera (la etapa 0), se conectan los nodos de la etapa 0 con los de la etapa 1.

  - Si la etapa es la penúltima (la etapa *numEtapas* - 2), se establece la conexión entre los nodos de la penúltima etapa y los nodos de la última etapa.

  - Para las etapas intermedias, se establecen las conexiones entre los nodos de la etapa en la que nos encontremos en el bucle y los nodos de la siguiente etapa. El número de conexiones y los nodos a los que se conectan se eligen de manera aleatoria.

  - Los valores de los arcos se generan aleatoriamente dentro del rango especificado por el usuario.

- Se transforma la matriz resultante a un *String* y se almacena en la posición *i* del *array* de datos.

- **obtenerNodosPorEtapa(int numNodos, int numEtapas):** se trata de un método privado que sirve para obtener en un mapa los nodos de cada etapa en un grafo por etapas.

Pulsa aquí para acceder al código

De forma resumida, el funcionamiento de este código es el siguiente:

- Se calcula el número de nodos intermedios (*numNodosIntermedios*) restando 2 al número total de nodos (*numNodos*).

- Se calcula el número de etapas intermedias (*numEtapasIntermedias*) restando 2 al número total de etapas (*numEtapas*).

- Se calcula el número de nodos por etapa (*numNodosPorEtapa*) dividiendo el número de nodos intermedios (*numNodosIntermedios*) entre el número de etapas intermedias (*numEtapasIntermedias*).

- Se calcula el número de nodos restantes (*numNodosRestantes*) obteniendo el resto de la división entre el número de nodos intermedios (*numNodosIntermedios*) entre el número de etapas intermedias (*numEtapasIntermedias*).

- Se inicializa la variable *nodoActual* a 1 y se crea un mapa (*mapaEtapaNodos*) para almacenar los nodos de cada etapa.

- Se recorren las etapas del grafo mediante un bucle for, de tal forma que:

  - Por cada etapa, se crea una lista de enteros que se corresponde con el listado de nodos de la etapa actual.

  - Si la etapa es la primera (la etapa 0), se agrega el nodo 0 a la lista de nodos de la etapa actual.

  - Si la etapa es la última (la etapa *numEtapas* – 1), se agrega el nodo *numNodos* – 1 a la lista de nodos de la etapa actual.

  - Para las etapas intermedias, se agrega un número determinado de nodos a la lista de nodos de la etapa actual.

    - El número de nodos se toma del valor de *numNodosPorEtapa*, y si hay nodos restantes, se agregan a la etapa actual antes de pasar a la siguiente.

  - Se agrega la lista de nodos de cada etapa al mapa *mapaEtapaNodos* con la etapa como clave.

- Por último, se devuelve el mapa *mapaEtapaNodos*, el cual contiene el listado de nodos de cada etapa del grafo.

5.5. Almacenamiento de restricciones de los generadores

Como en el caso de los generadores de permutaciones y el de tareas se reutilizan los atributos ya existentes y no requieren el tratamiento de nuevas restricciones, no se ha modificado nada en el código en referencia al almacenamiento de dichas restricciones.

Por lo tanto, como funcionalidad extra, para los generadores de grafos he incluido también tanto la posibilidad de almacenar las restricciones marcadas en la última experimentación como el hecho de poder guardar las restricciones en un fichero, de manera idéntica al resto de los generadores.

- Para lo primero, lo que hice fue reutilizar el atributo *TopesRecordados*, de tal forma que en el método *inicializarTopes* y en el método *inicializadorTopesConTiposIguales* (tanto en la clase *Aleatorios* como en la clase *Restricciones*), en el caso de la matriz de enteros (int[][]), se cambia el tamaño de *TopesRecordados[i]* de 4 a 15.

Pulsa aquí para acceder al código

- A la hora de guardar las restricciones en el fichero .cnstr, los valores que se almacenan en *TopesRecordados* en ese caso serían los siguientes:

- Los primeros cuatro elementos serían los típicos: tamaño de la primera dimensión, tamaño de la segunda dimensión, valor mínimo y valor máximo del rango de valores del grafo.

- El resto de los elementos serían los siguientes:

    - El quinto valor corresponde al del número de nodos del grafo.

    - El sexto valor se corresponde con la probabilidad de arco del grafo.

    - El séptimo valor corresponde al número de arcos del grafo.

    - El octavo valor corresponde al número de etapas del grafo.

    - El noveno valor tendrá diferentes valores, pues se corresponde a la dirección del grafo:

        - Si la dirección del grafo i no está almacenada, se le asigna la cadena “n/a”.

        - Si el grafo i es no dirigido, se le asigna la cadena “nodir”.

        - Si el grafo i es dirigido cíclico, se le asigna la cadena “dircic”.

        - Si el grafo i es dirigido acíclico, se le asigna la cadena “diracic”.

    - El décimo valor tendrá diferentes valores, pues se corresponde a si el grafo es valuado (o no):

        - Si la valuación del grafo i no está almacenada, se le asigna la cadena “n/a”.

        - Si el grafo i es valuado, se le asigna la cadena “val”.

        - Si el grafo i es no valuado, se le asigna la cadena “noval”.

    - El undécimo valor se corresponde al valor mínimo del rango de valores del grafo.

    - El duodécimo valor se corresponde al valor máximo del rango de valores del grafo.

    - El decimotercer valor tendrá diferentes valores, pues se corresponde a si el grafo es reflexivo (o no):

        - Si la reflexividad del grafo i no está almacenada, se le asigna la cadena “n/a”.

- ❖ Si el grafo i es reflexivo, se le asigna la cadena “ref”.

- ❖ Si el grafo i es no reflexivo, se le asigna la cadena “noref”.

- El decimocuarto valor tendrá diferentes valores, pues se corresponde a si el grafo es infinito en grafos (o no):

  - ❖ Si la opción de que el grafo i sea infinito en grafos no está almacenada, se le asigna la cadena “n/a”.

  - ❖ Si el grafo i es infinito en grafos, se le asigna la cadena “inf”.

  - ❖ Si el grafo i es no infinito en grafos, se le asigna la cadena “noinf”.

- El decimoquinto valor se corresponde al valor de los arcos no existentes del grafo a generar si éste es valuado.

Pulsa aquí para acceder al código

- En la clase *TXTImporter*, en los métodos *guardaRestriccionesTxt* (para guardar las restricciones recibidas en el fichero correspondiente) y *cargaRestricciones* (para cargar en la clase *Restricciones* las restricciones almacenadas en su fichero correspondiente).

  Como lo importante era tratar las restricciones almacenadas, modifiqué el método *cargaRestricciones*, haciendo lo siguiente:

  - Cuando se recorren las líneas del fichero, se obtiene un objeto de la clase *StringTokenizer* delimitado por espacios. Por lo que, si el número de tokens es igual a 15, se detecta que estamos tratando las restricciones de un grafo (es decir, una matriz de enteros).

  - Por lo tanto, si se cumple dicha condición, se almacenan los *Strings* en un *ArrayList* para que fueran más accesibles. Así pues, se va elemento a elemento (de la posición 4 a la 14) asignando los valores mencionados anteriormente a los atributos de la clase Restricciones, dependiendo en algunos casos de la cadena contenida en la posición i.

Pulsa aquí para acceder al código

5.6. Acceso a los generadores de datos

Para acceder a los generadores de datos se llama desde el método *aleatorioDatos* de la clase *PanelDatos* al método *tablaAleatoria*, bien de la clase *Aleatorios* como de la clase *Restricciones*, según el caso.

Pulsa aquí para acceder al código

Este método es el que hace que sean accesibles todos los generadores de datos, funcionando de la siguiente manera:

- En primer lugar, se llama al método *actualizarTopes* (en la clase *Restricciones*, el método *actualizaCotas*), almacenando el resultado en un entero al que llamamos *topesActualizados*.

- Dependiendo del valor de *topesActualizados*, se hacen unas acciones u otras:

    - Si el valor es igual a 0:

        - Se llama al método *tomaNumeroDatos* del *recordador* de la clase, en base al número de juegos de datos (*numArrows*).

        - Mediante un bucle for, se llama sucesivas veces al método *anadirFilaAleatoria*, correspondiente al generador de datos principal, para agregar una fila aleatoria a la tabla de datos.

        - Se llama al método *setCustomRowSorter* de la clase *PanelDatos* para establecer un orden personalizado de las filas en la tabla de datos.

        - Se crea una lista de enteros llamada *toReturn*, en la cual se insertan los valores 1 y 1 o 2 (dependiendo del caso).

            - El primer valor indica el generador de datos que se debe llamar en caso de que exista la fila en la tabla, y el segundo indica la versión (si es 1 es el de la clase Aleatorios y si es 2 es el de la clase Restricciones).

        - Por último, se devuelve la lista *toReturn* como resultado.

    - En el resto de los casos el proceso es similar, de tal forma que se diferencia en lo siguiente:

        - Si el valor es igual a 1:

            - Se llama al método *generarFilasTareas* en el bucle for.

            - En la lista a devolver se agrega el 2 (correspondiente al generador de tareas de duración no unitaria) y el valor correspondiente a la versión.

        - Si el valor es igual a 2:

            - Se llama al método *generarFilasPerms* en el bucle for.

- En la lista a devolver se agrega el 3 (correspondiente al generador de permutaciones) y el valor correspondiente a la versión.

- Si el valor es igual a 3:

  - Se llama al método *generarFilasGrafoProbs* en el bucle for.

  - En la lista a devolver se agrega el 4 (correspondiente al generador de grafos por probabilidad de arco) y el valor correspondiente a la versión.

- Si el valor es igual a 4:

  - Se llama al método *generarFilasGrafoArcos* en el bucle for.

  - En la lista a devolver se agrega el 5 (correspondiente al generador de grafos por el número de arcos) y el valor correspondiente a la versión.

- Si el valor es igual a 5:

  - Se llama al método *generarFilasGrafoEtapas* en el bucle for.

  - En la lista a devolver se agrega el 6 (correspondiente al generador de grafos por el número de etapas) y el valor correspondiente a la versión.

- Por último, si el valor de *topesActualizados* es distinto de todos los anteriores, se crea la lista de enteros *toReturn*, se le agrega un -1 (indicando que no se corresponde con un generador de datos o que se ha cancelado el diálogo del generador) y se devuelve la lista como resultado del método.

6. Evaluación

En este apartado de la memoria se explican las pruebas realizadas para comprobar el funcionamiento de lo implementado, además de las diferentes versiones que se han ido desarrollando con el paso del tiempo.

6.1. Pruebas y Evaluaciones de Usabilidad

En lo que se refiere a la realización de pruebas, no he seguido ningún procedimiento ni ninguna metodología en especial (como, por ejemplo, *Test Driven Development*).

Esto se debe a que la mayor parte de las mejoras desarrolladas están relacionadas con elementos de la interfaz gráfica de la aplicación y, por ende, deben ser probadas y testadas mediante diferentes pruebas directamente sobre el sistema AlgorEx.

Por lo tanto, para poder probar el funcionamiento de las diferentes mejoras y nuevas funcionalidades que se han ido implementando se han hecho diferentes pruebas con los ficheros proporcionados por el tutor, de manera progresiva.

Además, el tutor también ha ido evaluando informalmente las diferentes versiones desarrolladas que le he ido adjuntando en los diferentes correos electrónicos en los que me he comunicado con él.

Por otro lado, y con el fin de evaluar la usabilidad de la aplicación, se ha realizado un análisis heurístico, siguiendo la heurística de Shneiderman y sus ocho reglas de oro[14]:

1. **Esforzarse por la consistencia:** al tratarse de una aplicación asentada y desarrollada con anterioridad, las nuevas funcionalidades y cambios desarrollados en la aplicación se han hecho en base a la interfaz de la aplicación original, para mantener consistente la interfaz de usuario de la aplicación.

2. **Buscar la usabilidad universal:** aunque es una aplicación diseñada para un ámbito muy específico como es el de la experimentación de algoritmos, cualquier usuario relacionado con el mundo de la programación puede utilizarla sin ningún problema.

3. **Ofrecer comentarios informativos:** todas las acciones disponibles en la aplicación tienen su correspondiente respuesta, ya sea en forma de mensaje (al finalizar la acción), o bien al situar el ratón sobre el elemento que hace de accionador (lo que se conoce como *tooltip*). Por lo tanto, el usuario recibe *feedback* en cada acción realizada en la aplicación.

4. **Diseñar diálogos para producir un cierre:** todas las acciones para la generación de datos tienen su correspondiente diálogo, bien sea indicando éxito o un error, permitiendo dar por cerrado el proceso de generación de datos o el de ejecución de estos, según el caso.

5. **Prevenir errores:** la aplicación está diseñada para que en la generación de datos el usuario no cometa errores graves en la inserción de restricciones, de tal forma que si falta alguna restricción o alguna de ellas es incorrecta, se mostrará un diálogo con el mensaje de error correspondiente.

6. **Permitir la reversión de acciones:** como tal, muchas de las acciones no son reversibles, pero si, por ejemplo, hemos generado más juegos de datos que los que queríamos generar, es posible eliminar los datos de sesión y generar de nuevo el número de juegos de datos que teníamos en mente.

7. **Mantener a los usuarios en control:** debido a que no hay ninguna acción que resulte extraña o sea de difícil comprensión, los usuarios tendrán siempre la sensación de tener el control de la aplicación.

8. **Reducir la carga de la memoria a corto plazo:** como las restricciones de la generación de los juegos de datos se pueden almacenar y cargar en el experimento sin problema, los usuarios no tienen que recordar ningún tipo de información. Además, tanto los datos que se generan como los resultados de las ejecuciones están disponibles para el usuario en todo momento, por lo que tampoco deben memorizar esa información.

6.2. Versiones Desarrolladas

Pese a que he hecho uso de Git y GitHub como sistemas de control de versiones, el número de subidas al repositorio es bastante amplio y, en ocasiones, la subida de cambios al repositorio es por cambios mínimos y no porque sea una nueva versión.

Dado que a lo largo de estos meses me he puesto en contacto con el tutor mediante correo electrónico para enviarle los diferentes avances que iba realizando, resulta más efectivo analizar las diferentes versiones que he ido mandando al tutor en los diferentes correos que le he escrito en este tiempo.

Por lo tanto, las versiones desarrolladas a lo largo del desarrollo han sido las siguientes:

- **Versión del 30/10/2022:** en esta versión se solucionaba el problema de los juegos de datos repetidos en la generación aleatoria de datos (del generador principal).

- **Versión del 04/12/2022:** en esta versión se solventaba un problema que había con la generación de datos, ya que, al indicar, por ejemplo, 100 juegos de datos para un entero de 0 a 30 (en el ejemplo de Fibonacci), no mostraba un mensaje de advertencia y trataba de generar 100 juegos de datos sin repetición, entrando en un bucle infinito.

- **Versión del 19/02/2023:** en esta versión se incluye la ordenación de la tabla de ejecuciones y la sincronización de la ordenación entre la tabla de datos y la tabla de ejecuciones.

- **Versión del 07/03/2023:** en esta versión se incluye la posibilidad de introducir caracteres en los formatos de Java (incluyendo caracteres Unicode y secuencias de escape).

- **Versión del 15/03/2023:** en esta versión se resuelve un problema que surgía al cambiar de un fichero a otro en la experimentación, pues al cambiar de fichero no era posible generar juegos de datos, ya que no aparecía el diálogo correspondiente al generador de datos, sólo aparecía el diálogo en el que se solicita el número de juegos de datos a generar.

- **Versión del 30/03/2023:** en esta versión se incluye el cambio de sitio del botón de guardar restricciones.

- **Versión del 31/03/2023:** en esta versión se solventaba un pequeño fallo que había a la hora de generar juegos de datos, ya que cuando se encontraba con uno repetido, no se llamaba al método correcto para añadir las filas.

- **Versión del 10/04/2023:** en esta versión se incluía el generador de tareas de duración no unitaria y el generador de permutaciones (aún sin generalizar).

- **Versión del 28/04/2023:** en esta versión se incluye la interfaz de todos los generadores de grafos y el generador de grafos por probabilidad de arco funcional (sólo para casos en los que hubiera una matriz de enteros como único parámetro).

- **Versión del 14/05/2023:** en esta versión se incluyen todos los generadores de grafos funcionales, pero sólo cuando hubiera una matriz de enteros como único parámetro.

- **Versión del 05/06/2023:** en esta versión se incluyen todos los nuevos generadores de datos funcionales para todo tipo de casos. Además, se incluye la posibilidad de guardar restricciones para todos ellos.

- **Versión del 09/06/2023:** en esta versión se incluye la opción de Infinito en grafos para todos los generadores de grafos en caso de que los grafos a generar sean valuados.

- **Versión del 10/06/2023:** en esta versión se solventan algunos problemas que había, tales como que: la exportación de datos no se realizaba, por defecto, en un fichero TXT ; en el grafo multietapa los arcos no existentes se inicializaban a 0, en lugar de ser Short.MAX_VALUE ; al generar grafos valuados el valor por defecto para los arcos no existentes (en la opción de Infinito en grafos) era 1000, en lugar de ser Short.MAX_VALUE.

- **Versión del 07/07/2023:** se incluyen nuevos mensajes de error y se cambian el nombre de algunos métodos en el código.

7. Conclusiones y Trabajos Futuros

En este apartado de la memoria se muestran las conclusiones y las posibles mejoras que se podrían incluir en trabajos de fin de grado futuros en torno al sistema AlgorEx.

7.1. Conclusiones

Debido a que este trabajo de fin de grado se trata de un proyecto de actualización, en el que se le añaden mejoras y nuevas funcionalidades al sistema AlgorEx, el proceso de desarrollo me ha resultado ciertamente complejo.

Pese a la experiencia que he adquirido a lo largo de estos años en el grado en ingeniería informática, y en especial sobre Java, tuve que realizar un proceso de adaptación y aprendizaje sobre el proyecto, puesto que no era un proyecto que yo hubiese empezado de cero y que conociese al pie de la letra.

Debido a la gran cantidad de funcionalidades y el gran número de clases de las que se compone el proyecto, tuve que ir analizando una por una su significado y función, para poder desarrollar las mejoras y nuevas funcionalidades requeridas. Esto me ha supuesto un gran esfuerzo inicial, sobre todo para la familiarización y comprensión del código heredado.

Por otro lado, el aprendizaje que he obtenido en estos meses sobre las interfaces gráficas en Java ha sido muy gratificante, pues es una parte de Java de la que apenas conocía hasta que me puse a trabajar en este proyecto.

A pesar de lo costoso del proceso de desarrollo del proyecto, la experiencia que he adquirido en estos meses me empuja a seguir aprendiendo sobre Java y todo lo que rodea a este lenguaje de programación.

En referencia al esfuerzo realizado para la elaboración de este trabajo de fin de grado, a lo largo de esta memoria se han incluido:

- Las clases y métodos implementado, respectivamente un total de 12 clases y 38 métodos, teniendo algunos de ellos métodos equivalentes en otras clases (por ejemplo, el método *tablaAleatoria* que eran muy similares y tenían su versión en la clase *Aleatorios* y en la clase *Restricciones*).

- Las versiones desarrolladas y propuestas al tutor, llegando a un total de 14 versiones.

En lo que se refiere a reuniones con el tutor, éstas han sido reuniones puntuales para consultar dudas sobre la especificación y las nuevas funcionalidades que se iban requiriendo con el paso del tiempo y los avances realizados.

Por otro lado, el tiempo empleado para el desarrollo de este trabajo de fin de grado ha sido de unos nueve meses, pues comencé a principios de octubre de 2022 y estoy finalizando esta memoria a finales de julio de 2023.

Sin embargo, en los primeros meses (hasta enero), no tuve demasiado tiempo debido a que tenía una asignatura pendiente y estaba realizando las prácticas externas en empresa, por lo que no pude dedicarme de lleno a ello.

A partir de enero, tras la evaluación del primer cuatrimestre, me pude poner más tiempo con ello, dedicándole unas 7-8 horas diarias, ya que hasta abril estaba trabajando en horario de mañana y por las tardes me ponía con ello.

A partir de abril, pasé a trabajar a jornada completa y ese tiempo se redujo a unas 5-6 horas diarias. Evidentemente, los fines de semana le dedicaba más tiempo, llegando a estar unas 10-11 horas diarias para aprovechar al máximo.

7.2. Trabajos Futuros

De cara a posibles trabajos de fin de grado futuros en torno al sistema AlgorEx, la principal mejora que se debería incluir es una reestructuración del proyecto que fuese completa, puesto que está organizado en un único paquete (*optimex*) y dicho paquete está organizado en una cantidad inmensa de clases.

Sería bueno que se modularizara, no solo la estructura del proyecto, sino también las propias clases, puesto que hay clases como la clase Aleatorios o la clase Restricciones, cuyas funciones son prácticamente idénticas y, además, están compuestos por miles de líneas de código.

Por lo tanto, lo veo como un posible trabajo de fin de grado, ya que requeriría de muchas horas de trabajo e, incluso, sería factible el hecho de hacer una migración a un nuevo proyecto que fuese desarrollado de cero.

Quizá, de cara a un proyecto futuro, sería bueno también incluir versiones del sistema AlgorEx en más idiomas, pues por el momento sólo está en español y en inglés. Sería interesante, sobre todo, de cara a dar a conocer este proyecto en más países y que pudiera ser utilizado en buena parte de las universidades a lo largo y ancho del mundo, principalmente para el ámbito de la experimentación.

Bibliografía

[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/event/RowSorterListener.html

[2] https://docs.oracle.com/javase/8/docs/api/javax/swing/table/DefaultTableModel.html

[3] https://docs.oracle.com/javase/8/docs/api/javax/swing/table/AbstractTableModel.html

[4] https://www.manualweb.net/java/literales-java/

[5] https://docs.oracle.com/javase/8/docs/api/javax/swing/ButtonGroup.html

[6] https://www.javatpoint.com/java-actionlistener

[7] https://docs.oracle.com/javase/8/docs/api/javax/swing/Box.html

[8] https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

[9] https://www.jc-mouse.net/java/matriz-de-adyacencia-representacion-de-grafos-en-java

[10] 10.1002/cae.22423

[11] Su uso docente:
J. Ángel Velázquez Iturbide, "Using large-scale optimality testing as a tool for analysis tasks in algorithm courses", *2020 IEEE Frontiers in Education Conference - Proceedings*, IEEE Xplore, 2020, DOI 10.1109/FIE44824.2020.9273920

[12] Experimentación con algoritmos:
Catherine C. McGeoch, *A Guide to Experimental Algorithmics*, Cambridge University Press, 2012.

[13] Sartaj Sahni. *Data Structures, Algorithms, and Applications in Java*, 2ª ed. Silicon Press, 2004.

[14] https://worldcampus.saintleo.edu/noticias/cuales-son-las-ocho-reglas-de-oro-del-diseno-de-interfaces-de-ben-shneiderman

Anexo I – Códigos de Implementación

Método moverFila de la clase ModeloTabla:

```
public void moverFila(int desde, int hasta) {
    if (desde == hasta) {
        return;
    }
    Fila fila = datos.remove(desde);
    datos.add(hasta - (desde < hasta ? 1 : 0), fila);
    fireTableRowsDeleted(desde, desde);
    fireTableRowsInserted(hasta, hasta);
    fireTableRowsUpdated(Math.min(desde, hasta), Math.max(desde, hasta));
}
```

Método sorterChanged de la clase TableSync:

```
@Override
public void sorterChanged(RowSorterEvent e) {
    this.count++;

    if (this.count % 2 == 0) {
        List<Integer> commonElements = new LinkedList<>();
        for (int row = 0; row < table.getRowCount(); row++) {
            String element = table.getValueAt(row, columnIndex).toString();
            commonElements.add(Integer.parseInt(element));
        }

        ModeloTabla otherTableModel = (ModeloTabla) otherTable.getModel();
        for (int i = 0; i < commonElements.size(); i++) {
            for (int row = 0; row < otherTable.getRowCount(); row++) {
                int element = Integer.parseInt(otherTable.getValueAt(row, columnIndex).toString());
                if (element == commonElements.get(i)) {
                    otherTableModel.moverFila(row, i);
                    break;
                }
            }
        }
    }
}
```

Método getTableCellRendererComponent de la clase Modelo2RenderaColor:

```
public Component getTableCellRendererComponent(
    JTable table, Object obj, boolean isSelected, boolean hasFocus, int row, int column
) {

    Component cell = super.getTableCellRendererComponent(table, obj, isSelected, hasFocus, row, column);

    if (PanelDatos.tabla2.getRowSorter() == null) {
        ValueMethod = PanelDatos.RegisterMonitor.getMethodPos(row);
        PanelDatos.setCustomRowSorter(PanelDatos.modelo2, 2);
        PanelDatos.modeloDatos.getTable().getRowSorter().addRowSorterListener(
            new TableSync(PanelDatos.modeloDatos.getTable(), PanelDatos.modelo2.getTable(), 0)
        );
        PanelDatos.modelo2.getTable().getRowSorter().addRowSorterListener(
            new TableSync(PanelDatos.modelo2.getTable(), PanelDatos.modeloDatos.getTable(), 0)
        );
    } else
        ValueMethod = PanelDatos.RegisterMonitor.getMethodPos(
            PanelDatos.tabla2.getRowSorter().convertRowIndexToModel(row)
        );

    // Resto de la función ...
}
```

Método setCustomRowSorter de la clase PanelDatos:

```
public static void setCustomRowSorter(ModeloTabla modelo, int numTabla) {
    TableRowSorter<TableModel> sorter = new TableRowSorter<>(modelo);
    int i = 0;
    while (i < modelo.getTable().getColumnCount()) {
        if (numTabla == 1) {
            if (modelo.getTable().getColumnName(i).contains("int")
                    || modelo.getTable().getColumnName(i).contains("float")
                    || modelo.getTable().getColumnName(i).contains("double")
                    || modelo.getTable().getColumnName(i).contains("Núm.")
            ) {
                sorter.setComparator(i, new CustomComparator());
            }
        } else {
            sorter.setComparator(i, new CustomComparator());
        }
        i++;
    }
    modelo.getTable().setRowSorter(sorter);
}
```

Método compare de la clase CustomComparator:

```
@Override
public int compare(Object o1, Object o2) {
    String name1 = o1.toString();
    String name2 = o2.toString();

    if (name1.contains("=") && name2.contains("="))
        return name1.compareTo(name2);
    else if (name1.contains("="))
        name1 = name1.split("=")[0].trim();
    else if (name2.contains("="))
        name2 = name2.split("=")[0].trim();

    if (name1.startsWith("{") && name2.startsWith("{")) {
        name1 = name1.replace("{", "");
        name1 = name1.replace("}", "");
        name1 = name1.replace(",", " ");

        name2 = name2.replace("{", "");
        name2 = name2.replace("}", "");
        name2 = name2.replace(",", " ");

        name1 = name1.split(" ")[0];
        name2 = name2.split(" ")[0];
    }

    if (name1.contains(".") && name2.contains(".")) {
        float num1 = Parser.StringAFloat(name1);
        float num2 = Parser.StringAFloat(name2);
        return Float.compare(num1, num2);
    } else {
        int int1 = Parser.StringAEntero(name1);
        int int2 = Parser.StringAEntero(name2);
        return Integer.compare(int1, int2);
    }
}
```

Código del cambio de los atributos *charinf* y *charsup* en las clases Aleatorios y Restricciones:

```
private static String charinf = "\'0\'", charsup = "\'z\'";

public static void datosOptimalidad() {
    // Inicialización de datos ...
    charinf = "\'0\'";
    charsup = "\'z\'";
}

public static void datosEficiencia() {
    // Inicialización de datos ...
    charinf = "\'0\'";
    charsup = "\'z\'";
}
```

Método StringACaracter de la clase Parser:

```
public static char StringACaracter(String estrin) {
    estrin = estrin.trim();
    if (estrin.isEmpty()) {
        return ' ';
    }

    if (estrin.length() > 1) {
        int primerChar = estrin.indexOf('\'');
        int ultimoChar = estrin.lastIndexOf('\'');
        if (primerChar != -1 && ultimoChar != -1) {
            estrin = estrin.substring(1, estrin.length() - 1);
        } else {
            estrin = estrin.replace("'", "");
        }
    }

    char resultado = estrin.toCharArray()[0];
    if (resultado == '\\') {
        if (estrin.length() < 2) {
            return resultado;
        } else {
            char caracterEscapado = estrin.toCharArray()[1];
            switch (caracterEscapado) {
                case '\'':
                case '\"':
                case '\\':
                    resultado = estrin.toCharArray()[1];
                    break;
                case 'n':
                    resultado = '\n';
                    break;
                case 'r':
                    resultado = '\r';
                    break;
                case 't':
                    resultado = '\t';
                    break;
                case 'b':
                    resultado = '\b';
                    break;
                case 'f':
                    resultado = '\f';
                    break;
                case 'u':
                    if (estrin.length() < 6) {
                        resultado = ' ';
                    } else {
                        String cadenaUnicode = estrin.substring(2, 6);
                        resultado = (char) Integer.parseInt(cadenaUnicode, 16);
                    }
                    break;
                default:
```

```
                    resultado = ' ';
                }
            }
        }

        return resultado;
    }
```

Método anhadeFilaAleatoria de la clase ModeloTabla (versión anterior de AlgorEx):

```
public boolean anhadeFilaAleatoria(Fila nueva_fila, String oculto) {
    if (this.getTable().getRowCount() == 0) {
        id = 0;
    }
    conOcultos = true;
    boolean existe = PanelDatos.existeFila(nueva_fila);
    boolean filaCorrecta = esFilaCorrecta(nueva_fila);

    if (filaCorrecta) {
        if (!existe) {
            id++;
            nueva_fila.tomaCampoI(String.valueOf(id), 0);
            datos.add(nueva_fila);

            TableModelEvent evento;
            evento = new TableModelEvent(
                        this, this.getRowCount() - 1, this.getRowCount() - 1,
                        TableModelEvent.ALL_COLUMNS, TableModelEvent.INSERT
            );
            avisaSuscriptores(evento);
            datosOcultos.add(oculto);
        }
    }

    return (existe) || (!filaCorrecta);
}
```

Método anhadeFilaAleatoria de la clase ModeloTabla (versión actual de AlgorEx):

```
public boolean anhadeFilaAleatoria(Fila nueva_fila, String oculto, int opcion, int version) {
    if (this.getTable().getRowCount() == 0) {
        id = 0;
    }
    conOcultos = true;
    boolean existe = PanelDatos.existeFila(nueva_fila);
    boolean filaCorrecta = esFilaCorrecta(nueva_fila);

    if (filaCorrecta) {
        if (!existe) {
            id++;
            nueva_fila.tomaCampoI(String.valueOf(id), 0);
            datos.add(nueva_fila);

            TableModelEvent evento;

            evento = new TableModelEvent(
                this, this.getRowCount() - 1, this.getRowCount() - 1, TableModelEvent.ALL_COLUMNS,
                TableModelEvent.INSERT
            );
            avisaSuscriptores(evento);
            datosOcultos.add(oculto);
        } else {
            if (opcion == 1) {
                if (version == 1) {
                    Aleatorios.anadirFilaAleatoria();
                } else {
                    Restricciones.anadirFilaAleatoria();
                }
            } else if (opcion == 2) {
                if (version == 1) {
                    Aleatorios.generarFilasTareas();
```

```
                } else {
                    Restricciones.generarFilasTareas();
                }
            } else if (opcion == 3) {
                if (version == 1) {
                    Aleatorios.generarFilasPerms();
                } else {
                    Restricciones.generarFilasPerms();
                }
            } else if (opcion == 4) {
                if (version == 1) {
                    Aleatorios.generarFilasGrafoProbs();
                } else {
                    Restricciones.generarFilasGrafoProbs();
                }
            } else if (opcion == 5) {
                if (version == 1) {
                    Aleatorios.generarFilasGrafoArcos();
                } else {
                    Restricciones.generarFilasGrafoArcos();
                }
            } else if (opcion == 6) {
                if (version == 1) {
                    Aleatorios.generarFilasGrafoEtapas();
                } else {
                    Restricciones.generarFilasGrafoEtapas();
                }
            }
        }
    }

    return (existe) || (!filaCorrecta);
}
```

Método ventanaTareas de la clase Aleatorios:

```
private int ventanaTareas(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    // Ubicamos el lugar que ocupa cada array de enteros en el array de tipos para poder establecer los
    // topes
    List<Integer> posicionesArraysEnteros = IntStream.range(0, tipos.length)
            .filter(i -> tipos[i].equals("int[]"))
            .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);

    int posicionPrimerArray = posicionesArraysEnteros.get(0);
    int posicionSegundoArray = posicionesArraysEnteros.get(1);

    // Agregamos el panel correspondiente a los arrays de las tareas
    generarPanelTareas(tipos, posicionPrimerArray, posicionSegundoArray);

    // Recorremos los tipos para insertar los paneles correspondientes a el resto de parámetros (que no
    // sean int[])
    for (int i = 1; i < modeloDatos.getColumnCount(); i++) {
        if (!tipos[i].equals("int[]")) {
            generarPanelParametro(tipos, i);
        }
    }

    boolean tieneMatrizEnteros = Arrays.asList(tipos).contains("int[][]");

    if (tieneMatrizEnteros) {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 2) + 150));
    } else {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 3) + 80));
    }

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);
```

```
    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        int tamano = Parser.StringAEntero(TopesTextos[posicionPrimerArray][0].getText());
        if (tamano <= 0) {
            JOptionPane.showMessageDialog(null, mensaje20, mensaje13, JOptionPane.WARNING_MESSAGE);
            return -1;
        } else {
            // Array cs[]
            TopesReales[posicionPrimerArray][0] = TopesTextos[posicionPrimerArray][0].getText();
            TopesReales[posicionPrimerArray][1] = TopesTextos[posicionPrimerArray][0].getText();
            TopesRecordados[posicionPrimerArray][0] = TopesTextos[posicionPrimerArray][0].getText();
            TopesReales[posicionPrimerArray][2] = TopesTextos[posicionPrimerArray][1].getText();
            TopesRecordados[posicionPrimerArray][1] = TopesTextos[posicionPrimerArray][1].getText();
            TopesReales[posicionPrimerArray][3] = TopesTextos[posicionPrimerArray][2].getText();
            TopesRecordados[posicionPrimerArray][2] = TopesTextos[posicionPrimerArray][2].getText();

            // Array fs[]
            TopesReales[posicionSegundoArray][0] = TopesTextos[posicionPrimerArray][0].getText();
            TopesReales[posicionSegundoArray][1] = TopesTextos[posicionPrimerArray][0].getText();
            TopesRecordados[posicionSegundoArray][0] = TopesTextos[posicionPrimerArray][0].getText();
            TopesReales[posicionSegundoArray][2] = TopesTextos[posicionPrimerArray][1].getText();
            TopesRecordados[posicionSegundoArray][1] = TopesTextos[posicionPrimerArray][1].getText();
            TopesReales[posicionSegundoArray][3] = TopesTextos[posicionPrimerArray][2].getText();
            TopesRecordados[posicionSegundoArray][2] = TopesTextos[posicionPrimerArray][2].getText();

            // Resto de parámetros
            for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
                if (!tipos[i].equals("int[]")) {
                    establecerTopesUsuario(tipos, i);
                }
            }

            recordador.tomaTipos(tipos);
            recordador.tomaTopes(TopesRecordados);

            return 1;
        }
    } else {
        return -1;
    }
}
```

Método ventanaTareas de la clase Restricciones:

```
private int ventanaTareas(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    // Ubicamos el lugar que ocupa cada array de enteros en el array de tipos para poder establecer los
    // topes
    List<Integer> posicionesArraysEnteros = IntStream.range(0, tipos.length)
            .filter(i -> tipos[i].equals("int[]"))
            .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);

    int posicionPrimerArray = posicionesArraysEnteros.get(0);
```

```
int posicionSegundoArray = posicionesArraysEnteros.get(1);

// Agregamos el panel correspondiente a los arrays de las tareas
generarPanelTareas(tipos, posicionPrimerArray, posicionSegundoArray);

// Recorremos los tipos para insertar los paneles correspondientes a el resto de parámetros (que no
// sean int[])
for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
    if (!tipos[i].equals("int[]")) {
        generarPanelParametro(tipos, i);
    }
}

boolean tieneMatrizEnteros = Arrays.asList(tipos).contains("int[][]");

if (tieneMatrizEnteros) {
    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 150));
} else {
    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 2) + 80));
}

etiquetaprueba = new JLabel("");
JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
panel2.add(etiquetaprueba);
selector.add(panel2);

ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
icon.setImage(resize);

int respuesta;
respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
        JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

while (respuesta == 2) {
    respuesta = guardarRestricciones(tipos);
    if (respuesta == -1) {
        respuesta = JOptionPane.showOptionDialog(
                null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
        );
    }
}

if (respuesta == JOptionPane.OK_OPTION) {
    int tamano = Parser.StringAEntero(TopesTextos[posicionPrimerArray][0].getText());
    if (tamano <= 0) {
        JOptionPane.showMessageDialog(null, mensaje20, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    } else {
        // Array cs[]
        TopesReales[posicionPrimerArray][0] = TopesTextos[posicionPrimerArray][0].getText();
        TopesReales[posicionPrimerArray][1] = TopesTextos[posicionPrimerArray][0].getText();
        TopesRecordados[posicionPrimerArray][0] = TopesTextos[posicionPrimerArray][0].getText();
        TopesReales[posicionPrimerArray][2] = TopesTextos[posicionPrimerArray][1].getText();
        TopesRecordados[posicionPrimerArray][1] = TopesTextos[posicionPrimerArray][1].getText();
        TopesReales[posicionPrimerArray][3] = TopesTextos[posicionPrimerArray][2].getText();
        TopesRecordados[posicionPrimerArray][2] = TopesTextos[posicionPrimerArray][2].getText();

        // Array fs[]
        TopesReales[posicionSegundoArray][0] = TopesTextos[posicionPrimerArray][0].getText();
        TopesReales[posicionSegundoArray][1] = TopesTextos[posicionPrimerArray][0].getText();
        TopesRecordados[posicionSegundoArray][0] = TopesTextos[posicionPrimerArray][0].getText();
        TopesReales[posicionSegundoArray][2] = TopesTextos[posicionPrimerArray][1].getText();
        TopesRecordados[posicionSegundoArray][1] = TopesTextos[posicionPrimerArray][1].getText();
        TopesReales[posicionSegundoArray][3] = TopesTextos[posicionPrimerArray][2].getText();
        TopesRecordados[posicionSegundoArray][2] = TopesTextos[posicionPrimerArray][2].getText();

        // Resto de parámetros
        for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
            if (!tipos[i].equals("int[]")) {
                establecerTopesUsuario(tipos, i);
            }
```

```
            }

            recuerdacota.setTipos(tipos);
            recuerdacota.setCotas(TopesRecordados);

            return 1;
        }
    } else {
        return -1;
    }
}
```

Método generarPanelTareas de la clase Aleatorios:

```
private void generarPanelTareas(String[] tipos, int posicionPrimerArray, int posicionSegundoArray) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(
            tipos[posicionPrimerArray] + " " + Editor.parameterNames[posicionPrimerArray - 1] + ", " +
            tipos[posicionSegundoArray] + " " + Editor.parameterNames[posicionSegundoArray - 1]
    );
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    JLabel puntos = new JLabel(" ... ");
    puntos.setAlignmentX(Component.LEFT_ALIGNMENT);
    Label labelTamano = new Label(mensaje7);
    labelTamano.setAlignment(Label.LEFT);
    Label labelRango = new Label(mensaje6 + tipos[posicionPrimerArray].replace("[]", "") + ":");
    labelRango.setAlignment(Label.LEFT);

    TopesTextos[posicionPrimerArray][0].setAlignmentX(Component.LEFT_ALIGNMENT);
    TopesTextos[posicionPrimerArray][1].setAlignmentX(Component.LEFT_ALIGNMENT);
    TopesTextos[posicionPrimerArray][2].setAlignmentX(Component.LEFT_ALIGNMENT);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelTamano);
    panelinino1.add(TopesTextos[posicionPrimerArray][0]);
    panelinino1.add(Box.createHorizontalStrut(50));
    panelinino1.add(labelRango);
    panelinino1.add(TopesTextos[posicionPrimerArray][1]);
    panelinino1.add(puntos);
    panelinino1.add(TopesTextos[posicionPrimerArray][2]);
    panelinino1.add(Box.createHorizontalStrut(10));

    panelito.add(panelinino1);

    selector.add(panelito);
}
```

Método generarPanelTareas de la clase Restricciones:

```
private void generarPanelTareas(String[] tipos, int posicionPrimerArray, int posicionSegundoArray) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(
            tipos[posicionPrimerArray] + " " + Editor.parameterNames[posicionPrimerArray] + ", " +
            tipos[posicionSegundoArray] + " " + Editor.parameterNames[posicionSegundoArray]
    );
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    JLabel puntos = new JLabel(" ... ");
    puntos.setAlignmentX(Component.LEFT_ALIGNMENT);
    Label labelTamano = new Label(mensaje7);
    labelTamano.setAlignment(Label.LEFT);
    Label labelRango = new Label(mensaje6 + tipos[posicionPrimerArray].replace("[]", "") + ":");
    labelRango.setAlignment(Label.LEFT);

    TopesTextos[posicionPrimerArray][0].setAlignmentX(Component.LEFT_ALIGNMENT);
```

```
    TopesTextos[posicionPrimerArray][1].setAlignmentX(Component.LEFT_ALIGNMENT);
    TopesTextos[posicionPrimerArray][2].setAlignmentX(Component.LEFT_ALIGNMENT);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelTamano);
    panelinino1.add(TopesTextos[posicionPrimerArray][0]);
    panelinino1.add(Box.createHorizontalStrut(50));
    panelinino1.add(labelRango);
    panelinino1.add(TopesTextos[posicionPrimerArray][1]);
    panelinino1.add(puntos);
    panelinino1.add(TopesTextos[posicionPrimerArray][2]);
    panelinino1.add(Box.createHorizontalStrut(10));

    panelito.add(panelinino1);

    selector.add(panelito);
}
```

Método generarFilasTareas de la clase Aleatorios:

```
public static void generarFilasTareas() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    // Ubicamos el lugar que ocupa cada array de enteros en el array de tipos para poder modificar cada uno
    List<Integer> posicionesArraysEnteros = IntStream.range(0, tipos.length)
            .filter(i -> tipos[i].equals("int[]"))
            .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);

    int posicionPrimerArray = posicionesArraysEnteros.get(0);
    int posicionSegundoArray = posicionesArraysEnteros.get(1);

    // Asignamos las tareas a cada uno de los arrays, siendo el menor valor para cs (primer array) y el
    // mayor para fs (segundo array)
    int topeTareas = Parser.StringAEntero(TopesTextos[posicionPrimerArray][0].getText());
    int[] cs = new int[topeTareas];
    int[] fs = new int[topeTareas];
    asignarValoresTareas(posicionPrimerArray, topeTareas, cs, fs);

    // Asignamos los arrays a su correspondiente posición en el array de la tabla de datos
    datos[posicionPrimerArray] = Parser.ArrayEnteroAString(cs);
    datos[posicionSegundoArray] = Parser.ArrayEnteroAString(fs);

    // Generamos los datos para el resto de parámetros
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (!tipos[i].equals("int[]")) {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    // Generamos la fila con los datos y la añadimos a la tabla de datos
    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 2, 1);
}
```

Método generarFilasTareas de la clase Restricciones:

```
public static void generarFilasTareas() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    // Ubicamos el lugar que ocupa cada array de enteros en el array de tipos para poder modificar cada uno
```

```
    List<Integer> posicionesArraysEnteros = IntStream.range(0, tipos.length)
            .filter(i -> tipos[i].equals("int[]"))
            .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);

    int posicionPrimerArray = posicionesArraysEnteros.get(0);
    int posicionSegundoArray = posicionesArraysEnteros.get(1);

    // Asignamos las tareas a cada uno de los arrays, siendo el menor valor para cs (primer array) y el
    // mayor para fs (segundo array)
    int topeTareas = Parser.StringAEntero(TopesTextos[posicionPrimerArray - 1][0].getText());
    int[] cs = new int[topeTareas];
    int[] fs = new int[topeTareas];

    asignarValoresTareas(posicionPrimerArray, topeTareas, cs, fs);

    // Asignamos los arrays a su correspondiente posición en el array de la tabla de datos
    datos[posicionPrimerArray] = Parser.ArrayEnteroAString(cs);
    datos[posicionSegundoArray] = Parser.ArrayEnteroAString(fs);

    // Generamos los datos para el resto de parámetros
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (!tipos[i].equals("int[]")) {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    // Generamos la fila con los datos y la añadimos a la tabla de datos
    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 2, 2);
}
```

Método asignarValoresTareas de la clase Aleatorios y la clase Restricciones:

```
private static void asignarValoresTareas(int posicionPrimerArray, int topeTareas, int[] cs, int[] fs) {
    for (int i = 0; i < topeTareas; i++) {
        int entero1 = EnteroAleatorio(
                Parser.StringAEntero(TopesTextos[posicionPrimerArray][1].getText()),
                Parser.StringAEntero(TopesTextos[posicionPrimerArray][2].getText()));
        int entero2 = EnteroAleatorio(
                Parser.StringAEntero(TopesTextos[posicionPrimerArray][1].getText()),
                Parser.StringAEntero(TopesTextos[posicionPrimerArray][2].getText()));

        while (entero1 == entero2) {
            entero1 = EnteroAleatorio(
                    Parser.StringAEntero(TopesTextos[posicionPrimerArray][1].getText()),
                    Parser.StringAEntero(TopesTextos[posicionPrimerArray][2].getText()));
            entero2 = EnteroAleatorio(
                    Parser.StringAEntero(TopesTextos[posicionPrimerArray][1].getText()),
                    Parser.StringAEntero(TopesTextos[posicionPrimerArray][2].getText()));
        }

        cs[i] = Math.min(entero1, entero2);
        fs[i] = Math.max(entero1, entero2);
    }
}
```

Método ventanaPermutaciones de la clase Aleatorios:

```
private int ventanaPermutaciones(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 1; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[]")) {
            generarPanelPermutaciones(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
```

```
    }

    boolean tieneMatrizEnteros = Arrays.asList(tipos).contains("int[][]");

    if (tieneMatrizEnteros) {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 2) + 150));
    } else {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 2) + 105));
    }

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
            if (tipos[i].equals("int[]")) {
                int tope = Parser.StringAEntero(TopesTextos[i][0].getText());
                if (tope <= 0) {
                    JOptionPane.showMessageDialog(null, mensaje20, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                BigInteger combinaciones = factorial(tope);
                if (combinaciones.compareTo(BigInteger.valueOf(numArrows)) < 0) {
                    JOptionPane.showMessageDialog(null, mensaje14, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                } else {
                    int topeArray = Parser.StringAEntero(TopesTextos[i][0].getText()) - 1;
                    TopesReales[i][0] = TopesTextos[i][0].getText();
                    TopesReales[i][1] = TopesTextos[i][0].getText();
                    TopesRecordados[i][0] = TopesTextos[i][0].getText();
                    TopesReales[i][2] = "0";
                    TopesRecordados[i][1] = "0";
                    TopesReales[i][3] = Parser.EnteroAString(topeArray);
                    TopesRecordados[i][2] = Parser.EnteroAString(topeArray);
                }
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }
        recordador.tomaTipos(tipos);
        recordador.tomaTopes(TopesRecordados);
        return 2;
    } else {
        return -1;
    }
}
```

Método ventanaPermutaciones de la clase Restricciones:

```
private int ventanaPermutaciones(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("int[]")) {
            generarPanelPermutaciones(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    boolean tieneMatrizEnteros = Arrays.asList(tipos).contains("int[][]");

    if (tieneMatrizEnteros) {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 150));
    } else {
        selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 105));
    }

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
            if (tipos[i].equals("int[]")) {
                int tope = Parser.StringAEntero(TopesTextos[i][0].getText());
                if (tope <= 0) {
                    JOptionPane.showMessageDialog(null, mensaje20, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                BigInteger combinaciones = factorial(tope);
                if (combinaciones.compareTo(BigInteger.valueOf(numArrows)) < 0) {
                    JOptionPane.showMessageDialog(null, mensaje14, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                } else {
                    int topeArray = Parser.StringAEntero(TopesTextos[i][0].getText()) - 1;
                    TopesReales[i][0] = TopesTextos[i][0].getText();
                    TopesReales[i][1] = TopesTextos[i][0].getText();
                    TopesRecordados[i][0] = TopesTextos[i][0].getText();
                    TopesReales[i][2] = "0";
                    TopesRecordados[i][1] = "0";
                    TopesReales[i][3] = Parser.EnteroAString(topeArray);
                    TopesRecordados[i][2] = Parser.EnteroAString(topeArray);
                }
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }
        recuerdacota.setTipos(tipos);
```

```
        recuerdacota.setCotas(TopesRecordados);
        return 2;
    } else {
        return -1;
    }
}
```

Método generarPanelPermutaciones de la clase Aleatorios y Restricciones:

```
private void generarPanelPermutaciones(String[] tipos, int i) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(tipos[i] + " " + Editor.parameterNames[i - 1]);
    // En la clase Restricciones se usa Editor.parameterNames[i]
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    Label labelTamano = new Label(mensaje7);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalBox());
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelTamano);
    panelinino1.add(TopesTextos[i][0]);
    panelinino1.add(Box.createHorizontalStrut(325));

    panelito.add(panelinino1);

    selector.add(panelito);
}
```

Método generarFilasPerms de la clase Aleatorios:

```
public static void generarFilasPerms() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[]")) {
            int topePerms = Parser.StringAEntero(TopesTextos[i][0].getText());
            int[] perms = obtenerPermutaciones(topePerms);
            datos[i] = Parser.ArrayEnteroAString(perms);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 3, 1);
}
```

Método generarFilasPerms de la clase Restricciones:

```
public static void generarFilasPerms() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < tipos.length; i++) {
        if (tipos[i].equals("int[]")) {
            int topePerms = Parser.StringAEntero(TopesReales[i - 1][0].toString());
            int[] perms = obtenerPermutaciones(topePerms);
            datos[i] = Parser.ArrayEnteroAString(perms);
```

```
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 3, 2);
}
```

Método obtenerPermutaciones de la clase Aleatorios y la clase Restricciones:

```
private static int[] obtenerPermutaciones(int tope) {
    List<Integer> listaPerms = new LinkedList<>();
    for (int i = 0; i < tope; i++) {
        int enteroAleatorio = EnteroAleatorio(0, tope - 1);
        while (listaPerms.contains(enteroAleatorio)) {
            enteroAleatorio = EnteroAleatorio(0, tope - 1);
        }
        listaPerms.add(enteroAleatorio);
    }

    int[] arrayPerms = new int[tope];
    int i = 0;
    for (Integer perm : listaPerms) {
        arrayPerms[i] = perm;
        i++;
    }

    return arrayPerms;
}
```

Método factorial (tanto de la clase Aleatorios como de la clase Restricciones):

```
private BigInteger factorial(int n) {
    if (n == 0) {
        return BigInteger.valueOf(1);
    } else {
        return BigInteger.valueOf(n).multiply(factorial(n - 1));
    }
}
```

Código de los atributos de los grafos (de la clase Aleatorios y la clase Restricciones):

```
public static Map<Integer, JERoundTextField> textFieldsNodos;
private static String mensajeNodosGrafo;
public static Map<Integer, JERoundTextField> textFieldsProbabilidad;
private static String mensajeProbabilidadGrafo;
public static Map<Integer, JERoundTextField> textFieldsArcos;
private static String mensajeArcosGrafo;
public static Map<Integer, JERoundTextField> textFieldsEtapas;
private static String mensajeEtapasGrafo;
public static Map<Integer, boolean[]> direccionesGrafos;
private static String mensajeDireccionGrafo;
private static final String[] mensajesDireccionesGrafo = new String[3];
public static Map<Integer, Boolean> sonGrafosValuados;
private static Map<Integer, ButtonGroup> gruposBotonesValuado;
private static final String[] mensajesGrafoValuado = new String[3];
private static Map<Integer, JRadioButton> botonesSiValuado;
private static Map<Integer, JRadioButton> botonesNoValuado;
private static String mensajeRangoGrafo;
public static Map<Integer, JERoundTextField> textFieldsMinimo;
public static Map<Integer, JERoundTextField> textFieldsMaximo;
private static final String[] mensajesInfinitoEnGrafos = new String[3];
public static Map<Integer, Boolean> sonInfinitosEnGrafos;
private static Map<Integer, ButtonGroup> gruposBotonesValorInfinito;
private static String mensajeValorInfinito;
public static Map<Integer, JERoundTextField> textFieldsValorInfinito;
public static Map<Integer, Boolean> sonGrafosReflexivos;
private static Map<Integer, ButtonGroup> gruposBotonesReflexivo;
private static final String[] mensajesGrafoReflexivo = new String[3];
```

```
// ...
public static void inicializarDatosGrafos() {
    textFieldsNodos = new HashMap<>();
    textFieldsProbabilidad = new HashMap<>();
    textFieldsArcos = new HashMap<>();
    textFieldsEtapas = new HashMap<>();
    direccionesGrafos = new HashMap<>();
    sonGrafosValuados = new HashMap<>();
    gruposBotonesValuado = new HashMap<>();
    botonesSiValuado = new HashMap<>();
    botonesNoValuado = new HashMap<>();
    textFieldsMinimo = new HashMap<>();
    textFieldsMaximo = new HashMap<>();
    sonInfinitosEnGrafos = new HashMap<>();
    gruposBotonesValorInfinito = new HashMap<>();
    textFieldsValorInfinito = new HashMap<>();
    sonGrafosReflexivos = new HashMap<>();
    gruposBotonesReflexivo = new HashMap<>();
}
```

Método ventanaGrafoProbs de la clase Aleatorios:

```
private int ventanaGrafoProbs(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 1; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoProbs(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 170));

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);


    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
            if (tipos[i].equals("int[][]")) {
                if (textFieldsNodos.get(i).getText().equals("") ||
                        textFieldsProbabilidad.get(i).getText().equals("") ||
                        gruposBotonesValuado.get(i).getSelection() == null
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
```

```
                if (numNodos < 0) {
                    JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                float probabilidad = Parser.StringAFloat(textFieldsProbabilidad.get(i).getText());
                if (probabilidad < 0 || probabilidad > 1) {
                    JOptionPane.showMessageDialog(null, mensaje19, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (textFieldsMinimo.get(i).getText().equals("") ||
                        textFieldsMaximo.get(i).getText().equals(""))
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (gruposBotonesValorInfinito.get(i).getSelection() == null)
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (gruposBotonesValorInfinito.get(i).getSelection() != null) &&
                        (textFieldsValorInfinito.get(i).getText().equals(""))
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesNoValuado.get(i).isSelected() &&
                        gruposBotonesReflexivo.get(i).getSelection() == null
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }

        recordador.tomaTipos(tipos);
        recordador.tomaTopes(TopesRecordados);

        return 3;
    } else {
        return -1;
    }
}
```

Método ventanaGrafoProbs de la clase Restricciones:

```
private int ventanaGrafoProbs(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoProbs(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length) + 170));
```

```
etiquetaprueba = new JLabel("");
JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
panel2.add(etiquetaprueba);
selector.add(panel2);

ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
icon.setImage(resize);

int respuesta;
respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
        JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

while (respuesta == 2) {
    respuesta = guardarRestricciones(tipos);
    if (respuesta == -1) {
        respuesta = JOptionPane.showOptionDialog(
                null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
        );
    }
}

if (respuesta == JOptionPane.OK_OPTION) {
    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("int[][]")) {
            if (textFieldsNodos.get(i).getText().equals("") ||
                    textFieldsProbabilidad.get(i).getText().equals("") ||
                    gruposBotonesValuado.get(i).getSelection() == null
            ) {
                JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }

            int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
            if (numNodos < 0) {
                JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }

            float probabilidad = Parser.StringAFloat(textFieldsProbabilidad.get(i).getText());
            if (probabilidad < 0 || probabilidad > 1) {
                JOptionPane.showMessageDialog(null, mensaje19, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }


            if (botonesSiValuado.get(i).isSelected() &&
                    (textFieldsMinimo.get(i).getText().equals("") ||
                    textFieldsMaximo.get(i).getText().equals(""))
            ) {
                JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }

            if (botonesSiValuado.get(i).isSelected() &&
                    (gruposBotonesValorInfinito.get(i).getSelection() == null)
            ) {
                JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }

            if (botonesSiValuado.get(i).isSelected() &&
                    (gruposBotonesValorInfinito.get(i).getSelection() != null) &&
                    (textFieldsValorInfinito.get(i).getText().equals(""))
            ) {
                JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                return -1;
            }

            if (botonesNoValuado.get(i).isSelected() &&
                    gruposBotonesReflexivo.get(i).getSelection() == null
```

```
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }

        recuerdacota.setTipos(tipos);
        recuerdacota.setCotas(TopesRecordados);

        return 3;
    } else {
        return -1;
    }
}
```

Método generarPanelGrafoProbs de la clase Aleatorios y Restricciones:

```
private void generarPanelGrafoProbs(String[] tipos, int i) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(tipos[i] + " " + Editor.parameterNames[i - 1]);
    // En la clase Restricciones se usa Editor.parameterNames[i]
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    Label labelNodos = new Label(mensajeNodosGrafo);
    labelNodos.setAlignment(Label.LEFT);

    JERoundTextField textFieldNodos = textFieldsNodos.get(i);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelNodos);
    panelinino1.add(textFieldNodos);
    panelinino1.add(Box.createHorizontalStrut(340));

    panelito.add(panelinino1);

    Label labelProbabilidad = new Label(mensajeProbabilidadGrafo);
    labelProbabilidad.setAlignment(Label.LEFT);

    JERoundTextField textFieldProbabilidad = textFieldsProbabilidad.get(i);

    Box panelinino2 = Box.createHorizontalBox();
    panelinino2.add(Box.createVerticalStrut(20));
    panelinino2.add(Box.createHorizontalStrut(10));
    panelinino2.add(labelProbabilidad);
    panelinino2.add(textFieldProbabilidad);
    panelinino2.add(Box.createHorizontalStrut(165));

    panelito.add(panelinino2);

    Label labelDireccion = new Label(mensajeDireccionGrafo);
    labelDireccion.setAlignment(Label.LEFT);

    boolean[] direccionGrafo;
    if (direccionesGrafos.get(i) == null) {
        direccionGrafo = new boolean[]{true, false, false};
        direccionesGrafos.put(i, direccionGrafo);
    } else {
        direccionGrafo = direccionesGrafos.get(i);
    }

    JRadioButton botonNoDirigido = new JRadioButton(mensajesDireccionesGrafo[0]);
    botonNoDirigido.addActionListener(e -> {
        direccionGrafo[0] = true;
        direccionGrafo[1] = false;
```

```
    direccionGrafo[2] = false;
    direccionesGrafos.put(i, direccionGrafo);
});

JRadioButton botonDirigidoCiclico = new JRadioButton(mensajesDireccionesGrafo[1]);
botonDirigidoCiclico.addActionListener(e -> {
    direccionGrafo[0] = false;
    direccionGrafo[1] = true;
    direccionGrafo[2] = false;
    direccionesGrafos.put(i, direccionGrafo);
});

JRadioButton botonDirigidoAciclico = new JRadioButton(mensajesDireccionesGrafo[2]);
botonDirigidoAciclico.addActionListener(e -> {
    direccionGrafo[0] = false;
    direccionGrafo[1] = false;
    direccionGrafo[2] = true;
    direccionesGrafos.put(i, direccionGrafo);
});

ButtonGroup grupoOpcionesDireccion = new ButtonGroup();
grupoOpcionesDireccion.add(botonNoDirigido);
grupoOpcionesDireccion.add(botonDirigidoCiclico);
grupoOpcionesDireccion.add(botonDirigidoAciclico);
if (direccionesGrafos.get(i) == null) {
    grupoOpcionesDireccion.setSelected(botonNoDirigido.getModel(), true);
} else {
    if (direccionGrafo[0]) {
        grupoOpcionesDireccion.setSelected(botonNoDirigido.getModel(), true);
    } else if (direccionGrafo[1]) {
        grupoOpcionesDireccion.setSelected(botonDirigidoCiclico.getModel(), true);
    } else if (direccionGrafo[2]) {
        grupoOpcionesDireccion.setSelected(botonDirigidoAciclico.getModel(), true);
    }
}

Box panelinino3 = Box.createHorizontalBox();
panelinino3.add(Box.createVerticalStrut(20));
panelinino3.add(Box.createHorizontalStrut(10));
panelinino3.add(labelDireccion);
panelinino3.add(botonNoDirigido);
panelinino3.add(botonDirigidoCiclico);
panelinino3.add(botonDirigidoAciclico);
panelinino3.add(Box.createHorizontalStrut(70));

panelito.add(panelinino3);

Label labelValuado = new Label(mensajesGrafoValuado[0]);
labelValuado.setAlignment(Label.LEFT);

JRadioButton botonSiValuado = new JRadioButton(mensajesGrafoValuado[1]);
Box panelininoExtra = Box.createHorizontalBox();

Box panelininoExtra2 = Box.createHorizontalBox();
ButtonGroup grupoOpcionesInfinito = new ButtonGroup();
JRadioButton botonSiInfinito = new JRadioButton(mensajesInfinitoEnGrafos[1]);
JRadioButton botonNoInfinito = new JRadioButton(mensajesInfinitoEnGrafos[2]);
Box panelininoExtra3 = Box.createHorizontalBox();

botonSiValuado.addActionListener(e -> {
    panelininoExtra.removeAll();
    panelininoExtra.revalidate();

    panelininoExtra2.removeAll();
    panelininoExtra2.revalidate();

    panelininoExtra3.removeAll();
    panelininoExtra3.revalidate();

    grupoOpcionesInfinito.clearSelection();

    sonGrafosValuados.put(i, true);
    sonGrafosReflexivos.put(i, false);
```

```
    Label labelRango = new Label(mensajeRangoGrafo);
    labelRango.setAlignment(Label.LEFT);

    Label puntos = new Label("...");
    puntos.setAlignment(Label.LEFT);

    JERoundTextField textFieldMinimo = textFieldsMinimo.get(i);
    JERoundTextField textFieldMaximo = textFieldsMaximo.get(i);

    panelininoExtra.add(Box.createHorizontalStrut(10));
    panelininoExtra.add(labelRango);
    panelininoExtra.add(textFieldMinimo);
    panelininoExtra.add(Box.createHorizontalStrut(5));
    panelininoExtra.add(puntos);
    panelininoExtra.add(textFieldMaximo);
    panelininoExtra.add(Box.createHorizontalStrut(240));

    panelito.add(panelininoExtra);

    Label labelInfinitoEnGrafos = new Label(mensajesInfinitoEnGrafos[0]);
    labelInfinitoEnGrafos.setAlignment(Label.LEFT);

    botonSiInfinito.addActionListener(e1 -> {
        panelininoExtra3.removeAll();
        panelininoExtra3.revalidate();

        sonInfinitosEnGrafos.put(i, true);

        Label labelValorInfinito = new Label(mensajeValorInfinito);
        labelValorInfinito.setAlignment(Label.LEFT);

        JERoundTextField textFieldValorInfinito = textFieldsValorInfinito.get(i);

        panelininoExtra3.add(Box.createHorizontalStrut(10));
        panelininoExtra3.add(labelValorInfinito);
        panelininoExtra3.add(textFieldValorInfinito);
        panelininoExtra3.add(Box.createHorizontalStrut(50));

        panelito.add(panelininoExtra3);
        panelito.revalidate();
    });

    botonNoInfinito.addActionListener(e2 -> {
        panelininoExtra3.removeAll();
        panelininoExtra3.revalidate();

        sonInfinitosEnGrafos.put(i, false);
    });

    grupoOpcionesInfinito.add(botonSiInfinito);
    grupoOpcionesInfinito.add(botonNoInfinito);
    gruposBotonesValorInfinito.put(i, grupoOpcionesInfinito);

    panelininoExtra2.add(Box.createVerticalStrut(20));
    panelininoExtra2.add(Box.createHorizontalStrut(10));
    panelininoExtra2.add(labelInfinitoEnGrafos);
    panelininoExtra2.add(botonSiInfinito);
    panelininoExtra2.add(botonNoInfinito);
    panelininoExtra2.add(Box.createHorizontalStrut(270));

    panelito.add(panelininoExtra2);

    panelito.revalidate();
});
botonesSiValuado.put(i, botonSiValuado);

JRadioButton botonNoValuado = new JRadioButton(mensajesGrafoValuado[2]);
ButtonGroup grupoOpcionesReflexivo = new ButtonGroup();
JRadioButton botonSiReflexivo = new JRadioButton(mensajesGrafoReflexivo[1]);
JRadioButton botonNoReflexivo = new JRadioButton(mensajesGrafoReflexivo[2]);
botonNoValuado.addActionListener(e -> {
    panelininoExtra.removeAll();
```

```
        panelininoExtra.revalidate();

        panelininoExtra2.removeAll();
        panelininoExtra2.revalidate();

        panelininoExtra3.removeAll();
        panelininoExtra3.revalidate();

        grupoOpcionesReflexivo.clearSelection();

        sonGrafosValuados.put(i, false);
        sonInfinitosEnGrafos.put(i, false);

        Label labelReflexivo = new Label(mensajesGrafoReflexivo[0]);
        labelReflexivo.setAlignment(Label.LEFT);

        botonSiReflexivo.addActionListener(e1 -> sonGrafosReflexivos.put(i, true));
        botonNoReflexivo.addActionListener(e2 -> sonGrafosReflexivos.put(i, false));

        grupoOpcionesReflexivo.add(botonSiReflexivo);
        grupoOpcionesReflexivo.add(botonNoReflexivo);
        gruposBotonesReflexivo.put(i, grupoOpcionesReflexivo);

        panelininoExtra.add(Box.createVerticalStrut(20));
        panelininoExtra.add(Box.createHorizontalStrut(10));
        panelininoExtra.add(labelReflexivo);
        panelininoExtra.add(botonSiReflexivo);
        panelininoExtra.add(botonNoReflexivo);
        panelininoExtra.add(Box.createHorizontalStrut(330));

        panelito.add(panelininoExtra);
        panelito.revalidate();
    });
    botonesNoValuado.put(i, botonNoValuado);

    ButtonGroup grupoOpcionesValuado = new ButtonGroup();
    grupoOpcionesValuado.add(botonSiValuado);
    grupoOpcionesValuado.add(botonNoValuado);
    gruposBotonesValuado.put(i, grupoOpcionesValuado);

    Box panelinino4 = Box.createHorizontalBox();
    panelinino4.add(Box.createVerticalStrut(20));
    panelinino4.add(Box.createHorizontalStrut(10));
    panelinino4.add(labelValuado);
    panelinino4.add(botonSiValuado);
    panelinino4.add(botonNoValuado);
    panelinino4.add(Box.createHorizontalStrut(330));

    panelito.add(panelinino4);

    if (sonGrafosValuados.get(i) != null) {
        if (sonGrafosValuados.get(i)) {
            botonSiValuado.doClick();
            if (sonInfinitosEnGrafos.get(i) != null) {
                if (sonInfinitosEnGrafos.get(i)) {
                    botonSiInfinito.doClick();
                } else {
                    botonNoInfinito.doClick();
                }
            }
        } else {
            botonNoValuado.doClick();
            if (sonGrafosReflexivos.get(i) != null) {
                if (sonGrafosReflexivos.get(i)) {
                    botonSiReflexivo.doClick();
                } else {
                    botonNoReflexivo.doClick();
                }
            }
        }
    }
```

```
    selector.add(panelito);
}
```

Método generarFilasGrafoProbs de la clase Aleatorios:

```
public static void generarFilasGrafoProbs() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoProbs(datos, i);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 4, 1);
}
```

Método generarFilasGrafoProbs de la clase Restricciones:

```
public static void generarFilasGrafoProbs() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoProbs(datos, i - 1);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 4, 2);
}
```

Método generarDatosColumnaGrafoProbs de la clase Aleatorios y la clase Restricciones:

```
private static void generarDatosColumnaGrafoProbs(String[] datos, int i) {
    int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
    float probabilidad = Parser.StringAFloat(textFieldsProbabilidad.get(i).getText());
    int[][] grafo = new int[numNodos][numNodos];

    // Si el grafo es valuado, se inicializará el grafo de manera diferente, siendo un valor grande
    // (Short.MAX_VALUE o 1000, dependiendo de la elección del usuario) el valor en el caso en que
    // no haya arco entre un nodo u otro (salvo en la diagonal, que se queda como 0)
    boolean esGrafoValuado = sonGrafosValuados.get(i);
    boolean esInfinitoEnGrafo = sonInfinitosEnGrafos.get(i);
    if (esGrafoValuado) {
        if (esInfinitoEnGrafo) {
            int valorInfinito = Parser.StringAEntero(textFieldsValorInfinito.get(i).getText());
            for (int j = 0; j < numNodos; j++) {
                for (int k = j + 1; k < numNodos; k++) {
                    grafo[j][k] = valorInfinito;
                    grafo[k][j] = valorInfinito;
                }
            }
        } else {
            for (int j = 0; j < numNodos; j++) {
```

```
            for (int k = j + 1; k < numNodos; k++) {
                grafo[j][k] = Short.MAX_VALUE;
                grafo[k][j] = Short.MAX_VALUE;
            }
        }
    }
}

// Dependiendo de si es un grafo NO dirigido, un grafo dirigido cíclico, o un grafo dirigido acíclico,
// se asignan los valores de una manera u otra
boolean[] direccionGrafo = direccionesGrafos.get(i);
boolean esGrafoReflexivo = sonGrafosReflexivos.get(i);
if (direccionGrafo[0]) {
    if (!esGrafoValuado && esGrafoReflexivo) {
        for (int j = 0; j < numNodos; j++) {
            for (int k = 0; k < numNodos; k++) {
                float probabilidadAleatoria = FloatAleatorio(0, 1);
                if (probabilidad > probabilidadAleatoria) {
                    if (j <= k) {
                        grafo[j][k] = 1;
                        grafo[k][j] = 1;
                    }
                }
            }
        }
    } else {
        for (int j = 0; j < numNodos; j++) {
            for (int k = j + 1; k < numNodos; k++) {
                float probabilidadAleatoria = FloatAleatorio(0, 1);
                if (probabilidad > probabilidadAleatoria) {
                    if (esGrafoValuado) {
                        int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                        int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                        int valor = EnteroAleatorio(minimo, maximo);

                        grafo[j][k] = valor;
                        grafo[k][j] = valor;
                    } else {
                        grafo[j][k] = 1;
                        grafo[k][j] = 1;
                    }
                }
            }
        }
    }
} else if (direccionGrafo[1]) {
    for (int j = 0; j < numNodos; j++) {
        for (int k = 0; k < numNodos; k++) {
            float probabilidadAleatoria = FloatAleatorio(0, 1);
            if (probabilidad > probabilidadAleatoria) {
                if (j != k) {
                    if (esGrafoValuado) {
                        int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                        int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                        int valor = EnteroAleatorio(minimo, maximo);

                        grafo[j][k] = valor;
                    } else {
                        grafo[j][k] = 1;
                    }
                } else {
                    if (!esGrafoValuado && esGrafoReflexivo) {
                        grafo[j][k] = 1;
                    }
                }
            }
        }
    }
} else if (direccionGrafo[2]) {
    if (!esGrafoValuado && esGrafoReflexivo) {
        for (int j = 0; j < numNodos; j++) {
            for (int k = 0; k < numNodos; k++) {
                float probabilidadAleatoria = FloatAleatorio(0, 1);
```

```
                    if (probabilidad > probabilidadAleatoria) {
                        if (j <= k) {
                            grafo[j][k] = 1;
                        }
                    }
                }
            }
        } else {
            for (int j = 0; j < numNodos; j++) {
                for (int k = j + 1; k < numNodos; k++) {
                    float probabilidadAleatoria = FloatAleatorio(0, 1);
                    if (probabilidad > probabilidadAleatoria) {
                        if (esGrafoValuado) {
                            int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                            int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                            int valor = EnteroAleatorio(minimo, maximo);

                            grafo[j][k] = valor;
                        } else {
                            grafo[j][k] = 1;
                        }
                    }
                }
            }
        }
    }

    datos[i] = Parser.ArrayArrayEnteroAString(grafo); // En la clase Restricciones, en datos[i + 1]
}
```

Método ventanaGrafoArcos de la clase Aleatorios:

```
private int ventanaGrafoArcos(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 1; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoArcos(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 170));

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
```

```
if (tipos[i].equals("int[][]")) {
    if (textFieldsNodos.get(i).getText().equals("") ||
            textFieldsProbabilidad.get(i).getText().equals("") ||
            gruposBotonesValuado.get(i).getSelection() == null
    ) {
        JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    if (botonesSiValuado.get(i).isSelected() &&
            (textFieldsMinimo.get(i).getText().equals("") ||
            textFieldsMaximo.get(i).getText().equals(""))
    ) {
        JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    if (botonesSiValuado.get(i).isSelected() &&
            (gruposBotonesValorInfinito.get(i).getSelection() == null)
    ) {
        JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    if (botonesSiValuado.get(i).isSelected() &&
            (gruposBotonesValorInfinito.get(i).getSelection() != null) &&
            (textFieldsValorInfinito.get(i).getText().equals(""))
    ) {
        JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    if (botonesNoValuado.get(i).isSelected() &&
            gruposBotonesReflexivo.get(i).getSelection() == null
    ) {
        JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
    if (numNodos < 0) {
        JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
        return -1;
    }

    int numArcos = Parser.StringAEntero(textFieldsArcos.get(i).getText());
    if (direccionesGrafos.get(i)[1]) {
        int maxArcos;
        if (sonGrafosReflexivos.get(i)) {
            maxArcos = (int) Math.pow(numNodos, 2);
        } else {
            maxArcos = (int) Math.pow(numNodos, 2) - numNodos;
        }
        if (numArcos > maxArcos || numArcos < 0) {
            JOptionPane.showMessageDialog(
                    null, mensaje16 + maxArcos, mensaje13, JOptionPane.WARNING_MESSAGE
            );
            return -1;
        }
    } else if (direccionesGrafos.get(i)[0] || direccionesGrafos.get(i)[2]) {
        int maxArcos;
        if (sonGrafosReflexivos.get(i)) {
            maxArcos = (int) (Math.pow(numNodos, 2)) / 2;
        } else {
            maxArcos = (int) (Math.pow(numNodos, 2) - numNodos) / 2;
        }
        if (numArcos > maxArcos || numArcos < 0) {
            JOptionPane.showMessageDialog(
                    null, mensaje16 + maxArcos, mensaje13, JOptionPane.WARNING_MESSAGE
            );
            return -1;
        }
    }
```

```
                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }

        recordador.tomaTipos(tipos);
        recordador.tomaTopes(TopesRecordados);

        return 4;
    } else {
        return -1;
    }
}
```

Método ventanaGrafoArcos de la clase Restricciones:

```
private int ventanaGrafoArcos(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoArcos(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length) + 170));

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
            if (tipos[i].equals("int[][]")) {
                if (textFieldsNodos.get(i).getText().equals("") ||
                        textFieldsProbabilidad.get(i).getText().equals("") ||
                        gruposBotonesValuado.get(i).getSelection() == null
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (textFieldsMinimo.get(i).getText().equals("") ||
                        textFieldsMaximo.get(i).getText().equals(""))
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
```

```
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (gruposBotonesValorInfinito.get(i).getSelection() == null)
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesSiValuado.get(i).isSelected() &&
                        (gruposBotonesValorInfinito.get(i).getSelection() != null) &&
                        (textFieldsValorInfinito.get(i).getText().equals(""))
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                if (botonesNoValuado.get(i).isSelected() &&
                        gruposBotonesReflexivo.get(i).getSelection() == null
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
                if (numNodos < 0) {
                    JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                int numArcos = Parser.StringAEntero(textFieldsArcos.get(i).getText());
                if (direccionesGrafos.get(i)[1]) {
                    int maxArcos;
                    if (sonGrafosReflexivos.get(i)) {
                        maxArcos = (int) Math.pow(numNodos, 2);
                    } else {
                        maxArcos = (int) Math.pow(numNodos, 2) - numNodos;
                    }
                    if (numArcos > maxArcos || numArcos < 0) {
                        JOptionPane.showMessageDialog(
                                null, mensaje16 + maxArcos, mensaje13, JOptionPane.WARNING_MESSAGE
                        );
                        return -1;
                    }
                } else if (direccionesGrafos.get(i)[0] || direccionesGrafos.get(i)[2]) {
                    int maxArcos;
                    if (sonGrafosReflexivos.get(i)) {
                        maxArcos = (int) (Math.pow(numNodos, 2)) / 2;
                    } else {
                        maxArcos = (int) (Math.pow(numNodos, 2) - numNodos) / 2;
                    }
                    if (numArcos > maxArcos || numArcos < 0) {
                        JOptionPane.showMessageDialog(
                                null, mensaje16 + maxArcos, mensaje13, JOptionPane.WARNING_MESSAGE
                        );
                        return -1;
                    }
                }

                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }
        recuerdacota.setTipos(tipos);
        recuerdacota.setCotas(TopesRecordados);

        return 4;
    } else {
        return -1;
    }
}
```

Método generarPanelGrafoArcos de la clase Aleatorios y la clase Restricciones:

```
private void generarPanelGrafoArcos(String[] tipos, int i) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(tipos[i] + " " + Editor.parameterNames[i - 1]);
    // En la clase Restricciones se usa Editor.parameterNames[i]
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    Label labelNodos = new Label(mensajeNodosGrafo);
    labelNodos.setAlignment(Label.LEFT);

    JERoundTextField textFieldNodos = textFieldsNodos.get(i);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelNodos);
    panelinino1.add(textFieldNodos);
    panelinino1.add(Box.createHorizontalStrut(340));

    panelito.add(panelinino1);

    Label labelArcos = new Label(mensajeArcosGrafo);
    labelArcos.setAlignment(Label.LEFT);

    JERoundTextField textFieldArcos = textFieldsArcos.get(i);

    Box panelinino2 = Box.createHorizontalBox();
    panelinino2.add(Box.createVerticalStrut(20));
    panelinino2.add(Box.createHorizontalStrut(10));
    panelinino2.add(labelArcos);
    panelinino2.add(textFieldArcos);
    panelinino2.add(Box.createHorizontalStrut(340));

    panelito.add(panelinino2);

    Label labelDireccion = new Label(mensajeDireccionGrafo);
    labelDireccion.setAlignment(Label.LEFT);

    boolean[] direccionGrafo;
    if (direccionesGrafos.get(i) == null) {
        direccionGrafo = new boolean[]{true, false, false};
        direccionesGrafos.put(i, direccionGrafo);
    } else {
        direccionGrafo = direccionesGrafos.get(i);
    }

    JRadioButton botonNoDirigido = new JRadioButton(mensajesDireccionesGrafo[0]);
    botonNoDirigido.addActionListener(e -> {
        direccionGrafo[0] = true;
        direccionGrafo[1] = false;
        direccionGrafo[2] = false;
        direccionesGrafos.put(i, direccionGrafo);
    });

    JRadioButton botonDirigidoCiclico = new JRadioButton(mensajesDireccionesGrafo[1]);
    botonDirigidoCiclico.addActionListener(e -> {
        direccionGrafo[0] = false;
        direccionGrafo[1] = true;
        direccionGrafo[2] = false;
        direccionesGrafos.put(i, direccionGrafo);
    });

    JRadioButton botonDirigidoAciclico = new JRadioButton(mensajesDireccionesGrafo[2]);
    botonDirigidoAciclico.addActionListener(e -> {
        direccionGrafo[0] = false;
        direccionGrafo[1] = false;
        direccionGrafo[2] = true;
        direccionesGrafos.put(i, direccionGrafo);
    });
```

```
ButtonGroup grupoOpcionesDireccion = new ButtonGroup();
grupoOpcionesDireccion.add(botonNoDirigido);
grupoOpcionesDireccion.add(botonDirigidoCiclico);
grupoOpcionesDireccion.add(botonDirigidoAciclico);
if (direccionesGrafos.get(i) == null) {
    grupoOpcionesDireccion.setSelected(botonNoDirigido.getModel(), true);
} else {
    if (direccionGrafo[0]) {
        grupoOpcionesDireccion.setSelected(botonNoDirigido.getModel(), true);
    } else if (direccionGrafo[1]) {
        grupoOpcionesDireccion.setSelected(botonDirigidoCiclico.getModel(), true);
    } else if (direccionGrafo[2]) {
        grupoOpcionesDireccion.setSelected(botonDirigidoAciclico.getModel(), true);
    }
}

Box panelinino3 = Box.createHorizontalBox();
panelinino3.add(Box.createVerticalStrut(20));
panelinino3.add(Box.createHorizontalStrut(10));
panelinino3.add(labelDireccion);
panelinino3.add(botonNoDirigido);
panelinino3.add(botonDirigidoCiclico);
panelinino3.add(botonDirigidoAciclico);
panelinino3.add(Box.createHorizontalStrut(70));

panelito.add(panelinino3);

Label labelValuado = new Label(mensajesGrafoValuado[0]);
labelValuado.setAlignment(Label.LEFT);

JRadioButton botonSiValuado = new JRadioButton(mensajesGrafoValuado[1]);
Box panelininoExtra = Box.createHorizontalBox();

Box panelininoExtra2 = Box.createHorizontalBox();
ButtonGroup grupoOpcionesInfinito = new ButtonGroup();
JRadioButton botonSiInfinito = new JRadioButton(mensajesInfinitoEnGrafos[1]);
JRadioButton botonNoInfinito = new JRadioButton(mensajesInfinitoEnGrafos[2]);
Box panelininoExtra3 = Box.createHorizontalBox();

botonSiValuado.addActionListener(e -> {
    panelininoExtra.removeAll();
    panelininoExtra.revalidate();

    panelininoExtra2.removeAll();
    panelininoExtra2.revalidate();

    panelininoExtra3.removeAll();
    panelininoExtra3.revalidate();

    grupoOpcionesInfinito.clearSelection();

    sonGrafosValuados.put(i, true);
    sonGrafosReflexivos.put(i, false);

    Label labelRango = new Label(mensajeRangoGrafo);
    labelRango.setAlignment(Label.LEFT);

    Label puntos = new Label("...");
    puntos.setAlignment(Label.LEFT);

    JERoundTextField textFieldMinimo = textFieldsMinimo.get(i);
    JERoundTextField textFieldMaximo = textFieldsMaximo.get(i);

    panelininoExtra.add(Box.createHorizontalStrut(10));
    panelininoExtra.add(labelRango);
    panelininoExtra.add(textFieldMinimo);
    panelininoExtra.add(Box.createHorizontalStrut(5));
    panelininoExtra.add(puntos);
    panelininoExtra.add(textFieldMaximo);
    panelininoExtra.add(Box.createHorizontalStrut(240));

    panelito.add(panelininoExtra);
```

```
    Label labelInfinitoEnGrafos = new Label(mensajesInfinitoEnGrafos[0]);
    labelInfinitoEnGrafos.setAlignment(Label.LEFT);

    botonSiInfinito.addActionListener(e1 -> {
        panelininoExtra3.removeAll();
        panelininoExtra3.revalidate();

        sonInfinitosEnGrafos.put(i, true);

        Label labelValorInfinito = new Label(mensajeValorInfinito);
        labelValorInfinito.setAlignment(Label.LEFT);

        JERoundTextField textFieldValorInfinito = textFieldsValorInfinito.get(i);

        panelininoExtra3.add(Box.createHorizontalStrut(10));
        panelininoExtra3.add(labelValorInfinito);
        panelininoExtra3.add(textFieldValorInfinito);
        panelininoExtra3.add(Box.createHorizontalStrut(50));

        panelito.add(panelininoExtra3);
        panelito.revalidate();
    });

    botonNoInfinito.addActionListener(e2 -> {
        panelininoExtra3.removeAll();
        panelininoExtra3.revalidate();

        sonInfinitosEnGrafos.put(i, false);
    });

    grupoOpcionesInfinito.add(botonSiInfinito);
    grupoOpcionesInfinito.add(botonNoInfinito);
    gruposBotonesValorInfinito.put(i, grupoOpcionesInfinito);

    panelininoExtra2.add(Box.createVerticalStrut(20));
    panelininoExtra2.add(Box.createHorizontalStrut(10));
    panelininoExtra2.add(labelInfinitoEnGrafos);
    panelininoExtra2.add(botonSiInfinito);
    panelininoExtra2.add(botonNoInfinito);
    panelininoExtra2.add(Box.createHorizontalStrut(270));

    panelito.add(panelininoExtra2);

    panelito.revalidate();
});
botonesSiValuado.put(i, botonSiValuado);

JRadioButton botonNoValuado = new JRadioButton(mensajesGrafoValuado[2]);
ButtonGroup grupoOpcionesReflexivo = new ButtonGroup();
JRadioButton botonSiReflexivo = new JRadioButton(mensajesGrafoReflexivo[1]);
JRadioButton botonNoReflexivo = new JRadioButton(mensajesGrafoReflexivo[2]);
botonNoValuado.addActionListener(e -> {
    panelininoExtra.removeAll();
    panelininoExtra.revalidate();

    panelininoExtra2.removeAll();
    panelininoExtra2.revalidate();

    panelininoExtra3.removeAll();
    panelininoExtra3.revalidate();

    grupoOpcionesReflexivo.clearSelection();

    sonGrafosValuados.put(i, false);
    sonInfinitosEnGrafos.put(i, false);

    Label labelReflexivo = new Label(mensajesGrafoReflexivo[0]);
    labelReflexivo.setAlignment(Label.LEFT);

    botonSiReflexivo.addActionListener(e1 -> sonGrafosReflexivos.put(i, true));
    botonNoReflexivo.addActionListener(e2 -> sonGrafosReflexivos.put(i, false));

    grupoOpcionesReflexivo.add(botonSiReflexivo);
```

```
        grupoOpcionesReflexivo.add(botonNoReflexivo);
        gruposBotonesReflexivo.put(i, grupoOpcionesReflexivo);

        panelininoExtra.add(Box.createVerticalStrut(20));
        panelininoExtra.add(Box.createHorizontalStrut(10));
        panelininoExtra.add(labelReflexivo);
        panelininoExtra.add(botonSiReflexivo);
        panelininoExtra.add(botonNoReflexivo);
        panelininoExtra.add(Box.createHorizontalStrut(330));

        panelito.add(panelininoExtra);
        panelito.revalidate();
    });
    botonesNoValuado.put(i, botonNoValuado);

    ButtonGroup grupoOpcionesValuado = new ButtonGroup();
    grupoOpcionesValuado.add(botonSiValuado);
    grupoOpcionesValuado.add(botonNoValuado);
    gruposBotonesValuado.put(i, grupoOpcionesValuado);

    Box panelinino4 = Box.createHorizontalBox();
    panelinino4.add(Box.createVerticalStrut(20));
    panelinino4.add(Box.createHorizontalStrut(10));
    panelinino4.add(labelValuado);
    panelinino4.add(botonSiValuado);
    panelinino4.add(botonNoValuado);
    panelinino4.add(Box.createHorizontalStrut(330));

    panelito.add(panelinino4);

    if (sonGrafosValuados.get(i) != null) {
        if (sonGrafosValuados.get(i)) {
            botonSiValuado.doClick();
            if (sonInfinitosEnGrafos.get(i) != null) {
                if (sonInfinitosEnGrafos.get(i)) {
                    botonSiInfinito.doClick();
                } else {
                    botonNoInfinito.doClick();
                }
            }
        } else {
            botonNoValuado.doClick();
            if (sonGrafosReflexivos.get(i) != null) {
                if (sonGrafosReflexivos.get(i)) {
                    botonSiReflexivo.doClick();
                } else {
                    botonNoReflexivo.doClick();
                }
            }
        }
    }

    selector.add(panelito);
}
```

Método generarFilasGrafoArcos de la clase Aleatorios:

```
public static void generarFilasGrafoArcos() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoArcos(datos, i);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }
```

```
    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 5, 1);
}
```

Método generarFilasGrafoArcos de la clase Restricciones:

```
public static void generarFilasGrafoArcos() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoArcos(datos, i - 1);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 5, 2);
}
```

Método generarDatosColumnaGrafoArcos de la clase Aleatorios y la clase Restricciones:

```
private static void generarDatosColumnaGrafoArcos(String[] datos, int i) {
    int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
    int numArcos = Parser.StringAEntero(textFieldsArcos.get(i).getText());

    boolean[] direccionGrafo = direccionesGrafos.get(i);
    boolean esGrafoValuado = sonGrafosValuados.get(i);
    boolean esGrafoReflexivo = sonGrafosReflexivos.get(i);

    int[] arcos = generarArcosGrafo(numNodos, numArcos, direccionGrafo, esGrafoReflexivo);

    int[][] grafo = new int[numNodos][numNodos];

    // Si el grafo es valuado, se inicializará el grafo de manera diferente, siendo un valor grande
    // (Short.MAX_VALUE o 1000, dependiendo de la elección del usuario) el valor en el caso en que
    // no haya arco entre un nodo u otro (salvo en la diagonal, que se queda como 0)
    boolean esInfinitoEnGrafo = sonInfinitosEnGrafos.get(i);
    if (esGrafoValuado) {
        if (esInfinitoEnGrafo) {
            int valorInfinito = Parser.StringAEntero(textFieldsValorInfinito.get(i).getText());
            for (int j = 0; j < numNodos; j++) {
                for (int k = j + 1; k < numNodos; k++) {
                    grafo[j][k] = valorInfinito;
                    grafo[k][j] = valorInfinito;
                }
            }
        } else {
            for (int j = 0; j < numNodos; j++) {
                for (int k = j + 1; k < numNodos; k++) {
                    grafo[j][k] = Short.MAX_VALUE;
                    grafo[k][j] = Short.MAX_VALUE;
                }
            }
        }
    }

    // Dependiendo de si es un grafo NO dirigido, un grafo dirigido cíclico, o un grafo dirigido acíclico,
    // se asignan los valores de una manera u otra
    if (direccionGrafo[0]) {
        for (int arco : arcos) {
            int fila = arco / numNodos;
            int columna = arco % numNodos;
```

```
            if (esGrafoValuado) {
                int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                int valor = EnteroAleatorio(minimo, maximo);

                grafo[fila][columna] = valor;
                grafo[columna][fila] = valor;
            } else {
                grafo[fila][columna] = 1;
                grafo[columna][fila] = 1;
            }
        }
    } else if (direccionGrafo[1]) {
        for (int arco : arcos) {
            int fila = arco / numNodos;
            int columna = arco % numNodos;

            if (esGrafoValuado) {
                int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                int valor = EnteroAleatorio(minimo, maximo);

                grafo[fila][columna] = valor;
            } else {
                grafo[fila][columna] = 1;
            }
        }
    } else if (direccionGrafo[2]) {
        for (int arco : arcos) {
            int fila = arco / numNodos;
            int columna = arco % numNodos;

            if (esGrafoValuado) {
                int minimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                int maximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                int valor = EnteroAleatorio(minimo, maximo);

                grafo[fila][columna] = valor;
            } else {
                grafo[fila][columna] = 1;
            }
        }
    }

    datos[i] = Parser.ArrayArrayEnteroAString(grafo); // En la clase Restricciones, en datos[i + 1]
}
```

Método generarArcosGrafo de la clase Aleatorios y la clase Restricciones:

```
private static int[] generarArcosGrafo(
    int numNodos, int numArcos, boolean[] direccionGrafo, boolean esGrafoReflexivo
) {
    List<Integer> listaArcos = new LinkedList<>();
    for (int j = 0; j < numArcos; j++) {
        int arcoAleatorio;

        do {
            arcoAleatorio = EnteroAleatorio(0, ((int) Math.pow(numNodos, 2) - 1));

            // Evitar arcos en la misma posición y columna si el grafo no es reflexivo
            if (!esGrafoReflexivo) {
                int fila = arcoAleatorio / numNodos;
                int columna = arcoAleatorio % numNodos;
                if (fila == columna) {
                    arcoAleatorio = -1;
                }
            }

            // Evitar arcos en la mitad triangular inferior para grafos no dirigidos y dirigidos acíclicos
            if (direccionGrafo[0] || direccionGrafo[2]) {
                int fila = arcoAleatorio / numNodos;
                int columna = arcoAleatorio % numNodos;
```

```
                if (fila > columna) {
                    arcoAleatorio = -1;
                }
            }
        } while (listaArcos.contains(arcoAleatorio) || arcoAleatorio == -1);

        listaArcos.add(arcoAleatorio);
    }

    int[] arcos = new int[numArcos];
    int j = 0;
    for (Integer arco : listaArcos) {
        arcos[j] = arco;
        j++;
    }
    return arcos;
}
```

Método ventanaGrafoEtapas de la clase Aleatorios:

```
private int ventanaGrafoEtapas(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 1; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoEtapas(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 2) + 200));

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
        for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
            if (tipos[i].equals("int[][]")) {
                if (textFieldsNodos.get(i).getText().equals("") ||
                        textFieldsEtapas.get(i).getText().equals("")
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                if (gruposBotonesValorInfinito.get(i).getSelection() == null) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
```

```
                if (gruposBotonesValorInfinito.get(i).getSelection() != null &&
                        textFieldsValorInfinito.get(i).getText().equals("")
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
                if (numNodos < 0) {
                    JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                int numEtapas = Parser.StringAEntero(textFieldsEtapas.get(i).getText());
                if (numEtapas <= 2 || numEtapas > numNodos) {
                    JOptionPane.showMessageDialog(null, mensaje18, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }

        recordador.tomaTipos(tipos);
        recordador.tomaTopes(TopesRecordados);
        return 5;
    } else {
        return -1;
    }
}
```

Método ventanaGrafoEtapas de la clase Restricciones:

```
private int ventanaGrafoEtapas(String[] tipos) {
    selector = new JPanel();
    selector.setLayout(new BoxLayout(selector, BoxLayout.Y_AXIS));

    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("int[][]")) {
            generarPanelGrafoEtapas(tipos, i);
        } else {
            generarPanelParametro(tipos, i);
        }
    }

    selector.setPreferredSize(new Dimension(500, 90 * (tipos.length - 1) + 200));

    etiquetaprueba = new JLabel("");
    JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    panel2.add(etiquetaprueba);
    selector.add(panel2);

    ImageIcon icon = new ImageIcon(PanelIconos.class.getResource("imagenes/datos_aleatorios.png"));
    Image resize = icon.getImage().getScaledInstance(50, 50, Image.SCALE_SMOOTH);
    icon.setImage(resize);

    int respuesta;
    respuesta = JOptionPane.showOptionDialog(null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]);

    while (respuesta == 2) {
        respuesta = guardarRestricciones(tipos);
        if (respuesta == -1) {
            respuesta = JOptionPane.showOptionDialog(
                    null, selector, mensaje1, JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, icon, botonesGuardar, botonesGuardar[0]
            );
        }
    }

    if (respuesta == JOptionPane.OK_OPTION) {
```

```
        for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
            if (tipos[i].equals("int[][]")) {
                if (textFieldsNodos.get(i).getText().equals("") ||
                        textFieldsEtapas.get(i).getText().equals("")
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                if (gruposBotonesValorInfinito.get(i).getSelection() == null) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                if (gruposBotonesValorInfinito.get(i).getSelection() != null &&
                        textFieldsValorInfinito.get(i).getText().equals("")
                ) {
                    JOptionPane.showMessageDialog(null, mensaje15, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
                if (numNodos < 0) {
                    JOptionPane.showMessageDialog(null, mensaje17, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }
                int numEtapas = Parser.StringAEntero(textFieldsEtapas.get(i).getText());
                if (numEtapas < 2 || numEtapas > numNodos) {
                    JOptionPane.showMessageDialog(null, mensaje18, mensaje13, JOptionPane.WARNING_MESSAGE);
                    return -1;
                }

                establecerTopesUsuario(tipos, i);
            } else {
                establecerTopesUsuario(tipos, i);
            }
        }

        recuerdacota.setTipos(tipos);
        recuerdacota.setCotas(TopesRecordados);
        return 5;
    } else {
        return -1;
    }
}
```

Método generarPanelGrafoEtapas de la clase Aleatorios y la clase Restricciones:

```
private void generarPanelGrafoEtapas(String[] tipos, int i) {
    JPanel panelito = new JPanel();
    panelito.setLayout(new BoxLayout(panelito, BoxLayout.Y_AXIS));
    TitledBorder border = new TitledBorder(tipos[i] + " " + Editor.parameterNames[i - 1]);
    // En la clase Restricciones se usa Editor.parameterNames[i]
    border.setTitleFont(border.getTitleFont().deriveFont(Font.BOLD));
    panelito.setBorder(border);

    Label labelNodos = new Label(mensajeNodosGrafo);
    labelNodos.setAlignment(Label.LEFT);

    JERoundTextField textFieldNodos = textFieldsNodos.get(i);

    Box panelinino1 = Box.createHorizontalBox();
    panelinino1.add(Box.createHorizontalStrut(10));
    panelinino1.add(labelNodos);
    panelinino1.add(textFieldNodos);
    panelinino1.add(Box.createHorizontalStrut(340));

    panelito.add(panelinino1);

    Label labelEtapas = new Label(mensajeEtapasGrafo);
    labelEtapas.setAlignment(Label.LEFT);

    JERoundTextField textFieldEtapas = textFieldsEtapas.get(i);

    Box panelinino2 = Box.createHorizontalBox();
```

```
panelinino2.add(Box.createVerticalStrut(20));
panelinino2.add(Box.createHorizontalStrut(10));
panelinino2.add(labelEtapas);
panelinino2.add(textFieldEtapas);
panelinino2.add(Box.createHorizontalStrut(340));

panelito.add(panelinino2);

Label labelRango = new Label(mensajeRangoGrafo);
labelRango.setAlignment(Label.LEFT);

Label puntos = new Label("...");
puntos.setAlignment(Label.LEFT);

JERoundTextField textFieldMinimo = textFieldsMinimo.get(i);
JERoundTextField textFieldMaximo = textFieldsMaximo.get(i);

Box panelinino3 = Box.createHorizontalBox();
panelinino3.add(Box.createHorizontalStrut(10));
panelinino3.add(labelRango);
panelinino3.add(textFieldMinimo);
panelinino3.add(Box.createHorizontalStrut(5));
panelinino3.add(puntos);
panelinino3.add(textFieldMaximo);
panelinino3.add(Box.createHorizontalStrut(240));

panelito.add(panelinino3);

Box panelinino4 = Box.createHorizontalBox();

Label labelInfinitoEnGrafos = new Label(mensajesInfinitoEnGrafos[0]);
labelInfinitoEnGrafos.setAlignment(Label.LEFT);

ButtonGroup grupoOpcionesInfinito = new ButtonGroup();
JRadioButton botonSiInfinito = new JRadioButton(mensajesInfinitoEnGrafos[1]);
JRadioButton botonNoInfinito = new JRadioButton(mensajesInfinitoEnGrafos[2]);
Box panelininoExtra = Box.createHorizontalBox();

botonSiInfinito.addActionListener(e -> {
    panelininoExtra.removeAll();
    panelininoExtra.revalidate();

    sonInfinitosEnGrafos.put(i, true);

    Label labelValorInfinito = new Label(mensajeValorInfinito);
    labelValorInfinito.setAlignment(Label.LEFT);

    JERoundTextField textFieldValorInfinito = textFieldsValorInfinito.get(i);

    panelininoExtra.add(Box.createHorizontalStrut(10));
    panelininoExtra.add(labelValorInfinito);
    panelininoExtra.add(textFieldValorInfinito);
    panelininoExtra.add(Box.createHorizontalStrut(50));

    panelito.add(panelininoExtra);
    panelito.revalidate();
});

botonNoInfinito.addActionListener(e -> {
    panelininoExtra.removeAll();
    panelininoExtra.revalidate();

    sonInfinitosEnGrafos.put(i, false);
});

grupoOpcionesInfinito.add(botonSiInfinito);
grupoOpcionesInfinito.add(botonNoInfinito);
gruposBotonesValorInfinito.put(i, grupoOpcionesInfinito);

panelinino4.add(Box.createVerticalStrut(20));
panelinino4.add(Box.createHorizontalStrut(10));
panelinino4.add(labelInfinitoEnGrafos);
panelinino4.add(botonSiInfinito);
```

```
    panelinino4.add(botonNoInfinito);
    panelinino4.add(Box.createHorizontalStrut(270));

    panelito.add(panelinino4);

    if (sonInfinitosEnGrafos.get(i) != null) {
        if (sonInfinitosEnGrafos.get(i)) {
            botonSiInfinito.doClick();
        } else {
            botonNoInfinito.doClick();
        }
    }

    selector.add(panelito);
}
```

Método generarFilasGrafoEtapas de la clase Aleatorios:

```
public static void generarFilasGrafoEtapas() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    // Obtenemos los tipos almacenados
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    // Generamos los datos de cada columna, dando especial atención al caso particular de la matriz de
    // adyacencia del grafo
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoEtapas(datos, i);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 6, 1);
}
```

Método generarFilasGrafoEtapas de la clase Restricciones:

```
public static void generarFilasGrafoEtapas() {
    String[] tipos = new String[modeloDatos.getColumnCount()];
    String[] datos = new String[modeloDatos.getColumnCount()];

    // Obtenemos los tipos almacenados
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        tipos[i] = modeloDatos.getColumnType(i);
    }

    // Generamos los datos de cada fila, dando especial atención al caso particular de la matriz de
    // adyacencia del grafo
    for (int i = 0; i < modeloDatos.getColumnCount(); i++) {
        if (tipos[i].equals("int[][]")) {
            generarDatosColumnaGrafoEtapas(datos, i - 1);
        } else {
            generarDatosAleatoriosColumna(tipos, datos, i);
        }
    }

    Fila fila = new Fila(datos);
    modeloDatos.anhadeFilaAleatoria(fila, "noEjecutada", 6, 2);
}
```

Método generarDatosColumnaGrafoEtapas de las clases Aleatorios y Restricciones:

```
private static void generarDatosColumnaGrafoEtapas(String[] datos, int i) {
    int numNodos = Parser.StringAEntero(textFieldsNodos.get(i).getText());
```

```
int numEtapas = Parser.StringAEntero(textFieldsEtapas.get(i).getText());

int[][] grafo = new int[numNodos][numNodos];

// Como el grafo es valuado, se inicializará el grafo de manera diferente, siendo un valor grande
// (Short.MAX_VALUE o 1000, dependiendo de la elección del usuario) el valor en el caso en que no
// haya arco entre un nodo u otro (salvo en la diagonal, que se queda como 0)
boolean esInfinitoEnGrafo = sonInfinitosEnGrafos.get(i);
if (esInfinitoEnGrafo) {
    int valorInfinito = Parser.StringAEntero(textFieldsValorInfinito.get(i).getText());
    for (int j = 0; j < numNodos; j++) {
        for (int k = j + 1; k < numNodos; k++) {
            grafo[j][k] = valorInfinito;
            grafo[k][j] = valorInfinito;
        }
    }
} else {
    for (int j = 0; j < numNodos; j++) {
        for (int k = j + 1; k < numNodos; k++) {
            grafo[j][k] = Short.MAX_VALUE;
            grafo[k][j] = Short.MAX_VALUE;
        }
    }
}

Map<Integer, List<Integer>> mapaEtapasNodos = obtenerNodosPorEtapa(numNodos, numEtapas);

for (Integer etapa : mapaEtapasNodos.keySet()) {
    if (etapa < numEtapas - 1) {
        if (etapa == 0) {
            int nodoEtapaCero = mapaEtapasNodos.get(etapa).get(0);
            List<Integer> nodosEtapaUno = mapaEtapasNodos.get(etapa + 1);

            for (Integer nodo : nodosEtapaUno) {
                int rangoMinimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                int rangoMaximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                int valorArco = EnteroAleatorio(rangoMinimo, rangoMaximo);
                grafo[nodoEtapaCero][nodo] = valorArco;
            }
        } else if (etapa == numEtapas - 2) {
            int nodoUltimaEtapa = mapaEtapasNodos.get(etapa + 1).get(0);
            List<Integer> nodosPenultimaEtapa = mapaEtapasNodos.get(etapa);

            for (Integer nodo : nodosPenultimaEtapa) {
                int rangoMinimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                int rangoMaximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                int valorArco = EnteroAleatorio(rangoMinimo, rangoMaximo);
                grafo[nodo][nodoUltimaEtapa] = valorArco;
            }
        } else {
            List<Integer> nodosEtapaActual = mapaEtapasNodos.get(etapa);
            List<Integer> nodosEtapaSiguiente = mapaEtapasNodos.get(etapa + 1);

            for (Integer nodoOrigen : nodosEtapaActual) {
                int numArcosNodo = EnteroAleatorio(1, nodosEtapaSiguiente.size());
                List<Integer> nodosConexion = new ArrayList<>();
                for (int j = 0; j < numArcosNodo; j++) {
                    int indiceNodoAConectar = EnteroAleatorio(0, nodosEtapaSiguiente.size() - 1);
                    int nodoAConectar = nodosEtapaSiguiente.get(indiceNodoAConectar);
                    while (nodosConexion.contains(nodoAConectar)) {
                        indiceNodoAConectar = EnteroAleatorio(0, nodosEtapaSiguiente.size() - 1);
                        nodoAConectar = nodosEtapaSiguiente.get(indiceNodoAConectar);
                    }

                    nodosConexion.add(nodoAConectar);
                }


                for (Integer nodoDestino : nodosConexion) {
                    int rangoMinimo = Parser.StringAEntero(textFieldsMinimo.get(i).getText());
                    int rangoMaximo = Parser.StringAEntero(textFieldsMaximo.get(i).getText());
                    int valorArco = EnteroAleatorio(rangoMinimo, rangoMaximo);
                    grafo[nodoOrigen][nodoDestino] = valorArco;
```

```
                    }
                }
            }
        }
    }

    datos[i] = Parser.ArrayArrayEnteroAString(grafo); // En la clase Restricciones, en datos[i + 1]
}
```

Método obtenerNodosPorEtapa de la clase Aleatorios y la clase Restricciones:

```
private static Map<Integer, List<Integer>> obtenerNodosPorEtapa(int numNodos, int numEtapas) {
    // Quitamos el primer y último nodo
    int numNodosIntermedios = numNodos - 2;

    // Quitamos la primera y última etapa
    int numEtapasIntermedias = numEtapas - 2;

    int numNodosPorEtapa = numNodosIntermedios / numEtapasIntermedias;
    int numNodosRestantes = numNodosIntermedios % numEtapasIntermedias;

    int nodoActual = 1;
    Map<Integer, List<Integer>> mapaEtapaNodos = new HashMap<>();
    for (int etapa = 0; etapa < numEtapas; etapa++) {
        List<Integer> nodosEtapaActual = new ArrayList<>();
        int numNodosEtapaActual;

        if (etapa == 0) {
            nodosEtapaActual.add(0);
        } else if (etapa == numEtapas - 1) {
            nodosEtapaActual.add(numNodos - 1);
        } else {
            numNodosEtapaActual = numNodosPorEtapa;
            if (numNodosRestantes > 0) {
                numNodosEtapaActual += numNodosRestantes;
                numNodosRestantes = 0;
            }

            for (int i = 0; i < numNodosEtapaActual; i++) {
                nodosEtapaActual.add(nodoActual);
                nodoActual++;
            }
        }

        mapaEtapaNodos.put(etapa, nodosEtapaActual);
    }

    return mapaEtapaNodos;
}
```

Método inicializarTopes e inicializarTopesConTiposIguales en el caso de la matriz de enteros (clase Aleatorios y Restricciones):

```
private void inicializadorTopes(int columnassinnum, String[] tipos) {
    for (int i = 0; i < columnassinnum - 1; i++) {
        if (tipos[i].equals("void")) {
            // No hacer nada
        }
        // ...
        } else if (tipos[i].equals("int[][]") | tipos[i].equals("Integer[][]")) {
            TopesReales[i] = new Object[6];
            TopesTextos[i] = new JERoundTextField[4];
            TopesRecordados[i] = new Object[15];
            TopesTipos[i] = new Object[4];
            TopesTipos[i][0] = tamanoMaxArray;
            TopesTipos[i][1] = tamanoMaxSubArray;
            TopesTipos[i][2] = 1;
            TopesTipos[i][3] = intsup;

            JERoundTextField textFieldNodos = new JERoundTextField();
```

```
            textFieldNodos.setText("12");
            textFieldsNodos.put(i, textFieldNodos);

            JERoundTextField textFieldArcos = new JERoundTextField();
            textFieldArcos.setText("12");
            textFieldsArcos.put(i, textFieldArcos);

            JERoundTextField textFieldEtapas = new JERoundTextField();
            textFieldEtapas.setText("12");
            textFieldsEtapas.put(i, textFieldEtapas);

            JERoundTextField textFieldProbabilidad = new JERoundTextField();
            textFieldProbabilidad.setText("0.5");
            textFieldsProbabilidad.put(i, textFieldProbabilidad);

            JERoundTextField textFieldMinimo = new JERoundTextField();
            textFieldMinimo.setText("1");
            textFieldsMinimo.put(i, textFieldMinimo);

            JERoundTextField textFieldMaximo = new JERoundTextField();
            textFieldMaximo.setText("12");
            textFieldsMaximo.put(i, textFieldMaximo);

            JERoundTextField textFieldValorInfinito = new JERoundTextField();
            textFieldValorInfinito.setText(Parser.EnteroAString(Short.MAX_VALUE));
            textFieldsValorInfinito.put(i, textFieldValorInfinito);
        }
        // ...
    }
}

private void inicializadorTopesConTiposIguales(String[] tipos) {
    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("void")) {
            // No hacer nada
        }
        // ...
        } else if (tipos[i].equals("int[][]") | tipos[i].equals("Integer[][]")) {
            TopesReales[i] = new Object[6];
            TopesTextos[i] = new JERoundTextField[4];
            TopesRecordados[i] = new Object[15];
            TopesTipos[i] = new Object[4];
            TopesTipos[i][0] = recuerdacota.getCotas()[i][0];
            TopesTipos[i][1] = recuerdacota.getCotas()[i][1];
            TopesTipos[i][2] = recuerdacota.getCotas()[i][2];
            TopesTipos[i][3] = recuerdacota.getCotas()[i][3];
        }
        // ...
    }
}
```

Fragmento del método guardarRestricciones (clase Aleatorios y Restricciones):

```
private int guardarRestricciones(String[] tipos) {
    for (int i = 0; i < modeloDatos.getColumnCount() - 1; i++) {
        if (tipos[i].equals("void")) {
            // No hacer nada
        }
        // ...
        } else if (tipos[i].equals("int[][]") | tipos[i].equals("Integer[][]")) {
            TopesReales[i][0] = TopesTextos[i][0].getText();
            TopesReales[i][1] = TopesTextos[i][0].getText();
            TopesRecordados[i][0] = TopesTextos[i][0].getText();
            TopesReales[i][2] = TopesTextos[i][1].getText();
            TopesReales[i][3] = TopesTextos[i][1].getText();
            TopesRecordados[i][1] = TopesTextos[i][1].getText();
            TopesReales[i][4] = TopesTextos[i][2].getText();
            TopesRecordados[i][2] = TopesTextos[i][2].getText();
            TopesReales[i][5] = TopesTextos[i][3].getText();
            TopesRecordados[i][3] = TopesTextos[i][3].getText();

            // Topes correspondientes a los parámetros de los nuevos generadores de grafos
```

```
            TopesRecordados[i][4] = textFieldsNodos.get(i).getText();
            TopesRecordados[i][5] = textFieldsProbabilidad.get(i).getText();
            TopesRecordados[i][6] = textFieldsArcos.get(i).getText();
            TopesRecordados[i][7] = textFieldsEtapas.get(i).getText();

            if (direccionesGrafos.get(i) == null) {
                TopesRecordados[i][8] = "n/a";      // Sin asignar
            } else if (direccionesGrafos.get(i)[0]) {
                TopesRecordados[i][8] = "nodir";    // Grafo No Dirigido
            } else if (direccionesGrafos.get(i)[1]) {
                TopesRecordados[i][8] = "dircic";   // Grafo Dirigido Cíclico
            } else if (direccionesGrafos.get(i)[2]) {
                TopesRecordados[i][8] = "diracic";  // Grafo Dirigido Acíclico
            }

            if (sonGrafosValuados.get(i) == null) {
                TopesRecordados[i][9] = "n/a";      // Sin asignar
            } else if (sonGrafosValuados.get(i)) {
                TopesRecordados[i][9] = "val";      // Grafo Valuado
            } else if (!sonGrafosValuados.get(i)) {
                TopesRecordados[i][9] = "noval";    // Grafo No Valuado
            }

            TopesRecordados[i][10] = textFieldsMinimo.get(i).getText();
            TopesRecordados[i][11] = textFieldsMaximo.get(i).getText();

            if (sonGrafosReflexivos.get(i) == null) {
                TopesRecordados[i][12] = "n/a";     // Sin asignar
            } else if (sonGrafosReflexivos.get(i)) {
                TopesRecordados[i][12] = "ref";     // Grafo Reflexivo
            } else if (!sonGrafosReflexivos.get(i)) {
                TopesRecordados[i][12] = "noref";   // Grafo No Reflexivo
            }

            if (sonInfinitosEnGrafos.get(i) == null) {
                TopesRecordados[i][13] = "n/a"; // Sin asignar
            } else if (sonInfinitosEnGrafos.get(i)) {
                TopesRecordados[i][13] = "inf";   // Infinito en Grafos
            } else if (!sonInfinitosEnGrafos.get(i)) {
                TopesRecordados[i][13] = "noinf";  // No Infinito en Grafos
            }

            TopesRecordados[i][14] = textFieldsValorInfinito.get(i).getText();
        }
        // ...
    }
    // Resto del código
}
```

Método cargaRestricciones de la clase TXTImporter:

```
public Integer cargaRestricciones() {
    int ret = -1;
    try {
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);
        String linea;
        StringTokenizer aux;

        int tam = 10;

        // Initialize order and repeated data structures
        if (Restricciones.orderOptions == null)
            Restricciones.orderOptions = new boolean[tam][3];

        if (Restricciones.prevOrderOptions == null)
            Restricciones.prevOrderOptions = new boolean[tam][3];

        if (Restricciones.repeatedData == null)
            Restricciones.repeatedData = new boolean[tam];

        if (Restricciones.prevRepeatedData == null)
```

```
    Restricciones.prevRepeatedData = new boolean[tam];

while ((linea = br.readLine()) != null) {
    aux = new StringTokenizer(linea, " ");
    int numeroColumnas = aux.countTokens();

    String[] tipos = new String[numeroColumnas];
    Object[][] cota = new Object[0][0];
    RestriccionesCotas restCota = new RestriccionesCotas(tipos, cota);

    for (int i = 0; i < numeroColumnas; i++) {
        tipos[i] = aux.nextToken();
    }
    restCota.setTipos(tipos);

    Restricciones restriccion = new Restricciones(restCota);
    cota = restriccion.inicializaCotas(tipos);
    int j = 0;
    while ((linea = br.readLine()) != null && !linea.equals(";")) {
        aux = new StringTokenizer(linea, " ");
        int num = aux.countTokens();
        boolean isArrayLine = linea.contains("sr") || linea.contains("nr");
        boolean isGraphLine = (num == 15);

        if (isArrayLine)
            num -= 2;

        Object[] objetos = new Object[num];
        String order = "";
        String rep = "";

        for (int i = 0; i < num; i++) {
            objetos[i] = aux.nextToken();
        }
        cota[j] = objetos;

        // Read order and repeated options
        if (isArrayLine) {
            order = aux.nextToken();
            if (Objects.equals(order, "alea")) {
                Restricciones.prevOrderOptions[j][0] = true;
                Restricciones.prevOrderOptions[j][1] = false;
                Restricciones.prevOrderOptions[j][2] = false;
            } else if (Objects.equals(order, "asc")) {
                Restricciones.prevOrderOptions[j][0] = false;
                Restricciones.prevOrderOptions[j][1] = true;
                Restricciones.prevOrderOptions[j][2] = false;
            } else if (Objects.equals(order, "desc")) {
                Restricciones.prevOrderOptions[j][0] = false;
                Restricciones.prevOrderOptions[j][1] = false;
                Restricciones.prevOrderOptions[j][2] = true;
            }

            rep = aux.nextToken();
            if (Objects.equals(rep, "sr")) {
                Restricciones.prevRepeatedData[j] = true;
            } else if (Objects.equals(rep, "nr")) {
                Restricciones.prevRepeatedData[j] = false;
            }
        }

        // Read graph options
        if (isGraphLine) {
            Restricciones.inicializarDatosGrafos();

            ArrayList<String> graphTokens = new ArrayList<>();
            aux = new StringTokenizer(linea, " ");
            while (aux.hasMoreTokens()) {
                String token = aux.nextToken();
                graphTokens.add(token);
            }

            JERoundTextField textFieldNodos = new JERoundTextField();
```

```
            textFieldNodos.setText(graphTokens.get(4));
            Restricciones.textFieldsNodos.put(j, textFieldNodos);

            JERoundTextField textFieldProbabilidad = new JERoundTextField();
            textFieldProbabilidad.setText(graphTokens.get(5));
            Restricciones.textFieldsProbabilidad.put(j, textFieldProbabilidad);

            JERoundTextField textFieldArcos = new JERoundTextField();
            textFieldArcos.setText(graphTokens.get(6));
            Restricciones.textFieldsArcos.put(j, textFieldArcos);

            JERoundTextField textFieldEtapas = new JERoundTextField();
            textFieldEtapas.setText(graphTokens.get(7));
            Restricciones.textFieldsEtapas.put(j, textFieldEtapas);

            if (!graphTokens.get(8).equals("n/a")) {
                boolean[] direccionesGrafo = new boolean[3];
                switch (graphTokens.get(8)) {
                    case "nodir":
                        direccionesGrafo[0] = true;
                        Restricciones.direccionesGrafos.put(j, direccionesGrafo);
                        break;
                    case "dircic":
                        direccionesGrafo[1] = true;
                        Restricciones.direccionesGrafos.put(j, direccionesGrafo);
                        break;
                    case "diracic":
                        direccionesGrafo[2] = true;
                        Restricciones.direccionesGrafos.put(j, direccionesGrafo);
                        break;
                }
            }

            if (!graphTokens.get(9).equals("n/a")) {
                if (graphTokens.get(9).equals("val")) {
                    Restricciones.sonGrafosValuados.put(j, true);
                } else if (graphTokens.get(9).equals("noval")) {
                    Restricciones.sonGrafosValuados.put(j, false);
                }
            }

            JERoundTextField textFieldValuadoMinimo = new JERoundTextField();
            textFieldValuadoMinimo.setText(graphTokens.get(10));
            Restricciones.textFieldsMinimo.put(j, textFieldValuadoMinimo);

            JERoundTextField textFieldValuadoMaximo = new JERoundTextField();
            textFieldValuadoMaximo.setText(graphTokens.get(11));
            Restricciones.textFieldsMaximo.put(j, textFieldValuadoMaximo);

            if (!graphTokens.get(12).equals("n/a")) {
                if (graphTokens.get(12).equals("ref")) {
                    Restricciones.sonGrafosReflexivos.put(j, true);
                } else if (graphTokens.get(12).equals("noref")) {
                    Restricciones.sonGrafosReflexivos.put(j, false);
                }
            }

            if (!graphTokens.get(13).equals("n/a")) {
                if (graphTokens.get(13).equals("inf")) {
                    Restricciones.sonInfinitosEnGrafos.put(j, true);
                } else {
                    Restricciones.sonInfinitosEnGrafos.put(j, false);
                }
            }

            JERoundTextField textFieldValorInfinito = new JERoundTextField();
            textFieldValorInfinito.setText(graphTokens.get(14));
            Restricciones.textFieldsValorInfinito.put(j, textFieldValorInfinito);
        }

        j++;
    }
    restCota.setCotas(cota);
```

```
            resimportadas.insertaRestriccion(restCota);
            ret = 0;
        }
    } catch (IOException e) {
        e.printStackTrace();
        return -1;
    } finally {
        try {
            if (null != fr) {
                fr.close();
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
    return ret;
}
```

Método tablaAleatoria de la clase Aleatorios:

```
public List<Integer> tablaAleatoria(int numArrows) {
    int topesActualizados = actualizarTopes();
    if (topesActualizados == 0) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            anadirFilaAleatoria();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(1);
        toReturn.add(1);
        return toReturn;
    } else if (topesActualizados == 1) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            generarFilasTareas();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(2);
        toReturn.add(1);
        return toReturn;
    } else if (topesActualizados == 2) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            generarFilasPerms();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(3);
        toReturn.add(1);
        return toReturn;
    } else if (topesActualizados == 3) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoProbs();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(4);
        toReturn.add(1);
        return toReturn;
    } else if (topesActualizados == 4) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoArcos();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(5);
        toReturn.add(1);
```

```
        return toReturn;
    } else if (topesActualizados == 5) {
        recordador.tomaNumeroDatos(numArrows);
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoEtapas();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(6);
        toReturn.add(1);
        return toReturn;
    } else {
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(-1);
        return toReturn;
    }
}
```

Método tablaAleatoria de la clase Restricciones:

```
public List<Integer> tablaAleatoria(int numArrows) {
    int topesActualizados = actualizaCotas();
    if (topesActualizados == 0) {
        for (int i = 0; i < numArrows; i++) {
            anadirFilaAleatoria();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(1);
        toReturn.add(2);
        return toReturn;
    } else if (topesActualizados == 1) {
        for (int i = 0; i < numArrows; i++) {
            generarFilasTareas();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(2);
        toReturn.add(2);
        return toReturn;
    } else if (topesActualizados == 2) {
        for (int i = 0; i < numArrows; i++) {
            generarFilasPerms();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(3);
        toReturn.add(2);
        return toReturn;
    } else if (topesActualizados == 3) {
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoProbs();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(4);
        toReturn.add(2);
        return toReturn;
    } else if (topesActualizados == 4) {
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoArcos();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(5);
        toReturn.add(2);
        return toReturn;
```

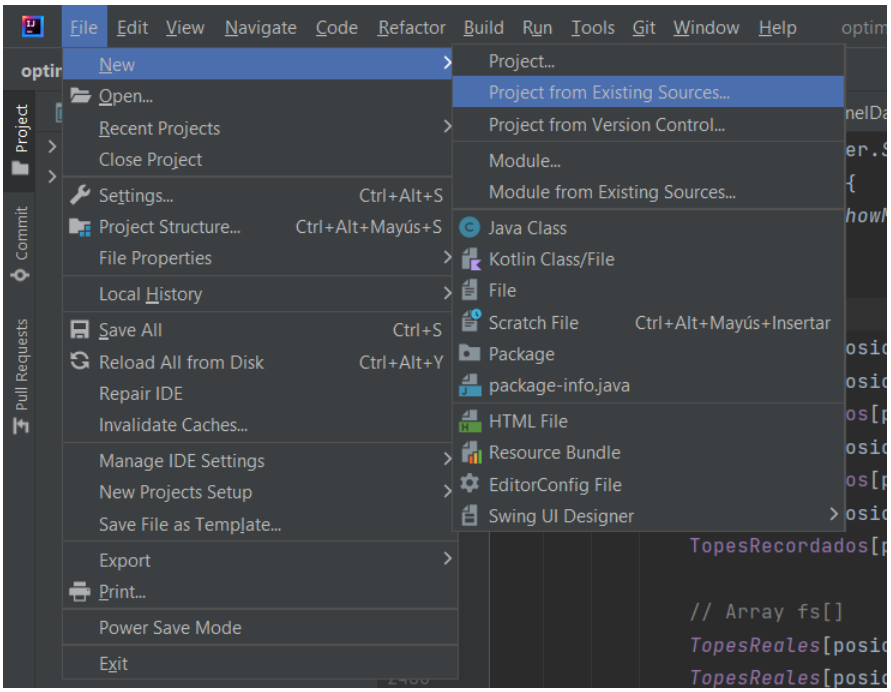
```
    } else if (topesActualizados == 5) {
        for (int i = 0; i < numArrows; i++) {
            generarFilasGrafoEtapas();
        }
        PanelDatos.setCustomRowSorter(PanelDatos.modeloDatos, 1);

        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(6);
        toReturn.add(2);
        return toReturn;
    } else {
        List<Integer> toReturn = new ArrayList<>();
        toReturn.add(-1);
        return toReturn;
    }
}
```

Anexo II – Configuración para ampliaciones futuras

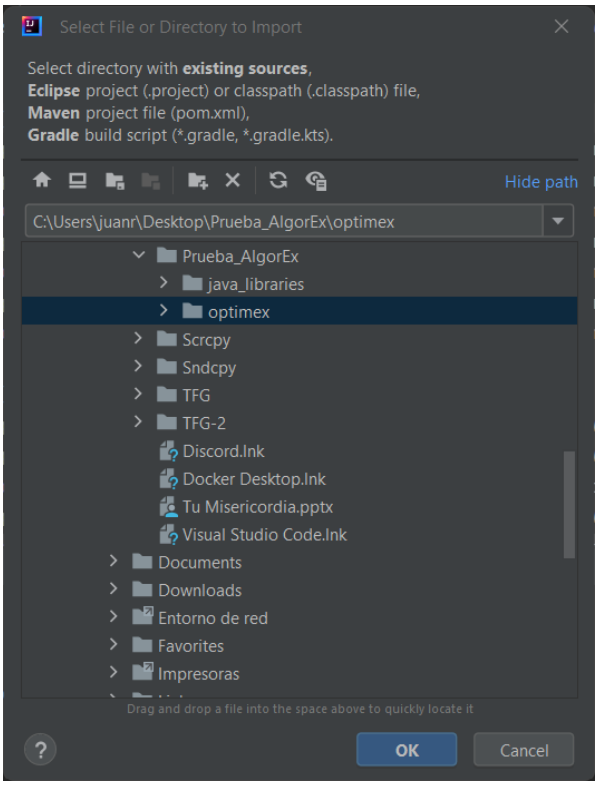
En este **apartado** de la **memoria** se incluyen los pasos necesarios para poder importar y poner en marcha el **sistema AlgorEx** en nuestro entorno de desarrollo, en este caso en ***IntelliJ Idea*.** Todo ello con el fin de dar soporte a aquellos **desarrolladores** que realicen ampliaciones o mejoras sobre el **sistema** en un futuro.

Para importar el **proyecto**, una vez tengamos guardadas en un **directorio** las carpetas **optimex** (con el contenido del proyecto) y **java_libraries** (contiene las librerías a utilizar), abrimos ***IntelliJ Idea*** y hacemos lo siguiente:

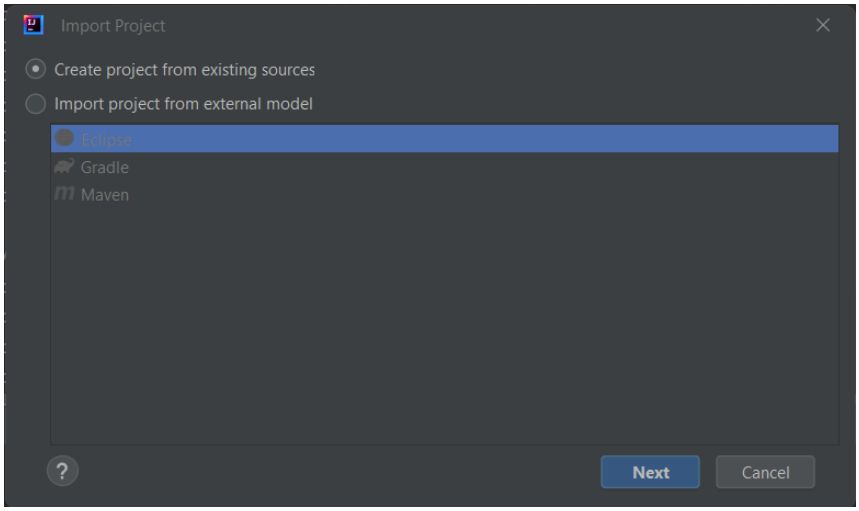
1. Una vez abierto ***IntelliJ Idea*** (habitualmente se abre con el último **proyecto** que tuviéramos abierto), seleccionamos **File → New → Project from Existing Sources…**



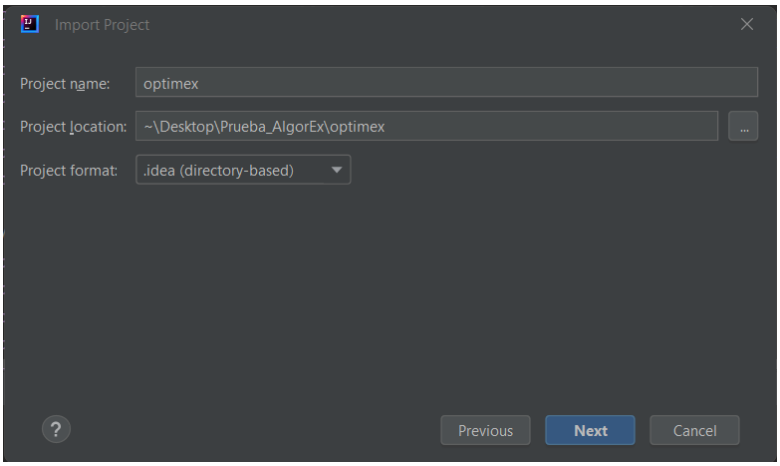
2. Se abre un **diálogo** que nos permite seleccionar el **proyecto** que queremos importar. Dentro del **directorio** donde se han guardado las carpetas **optimex** y **java_libraries** (Prueba_AlgorEx), se selecciona la carpeta **optimex**:



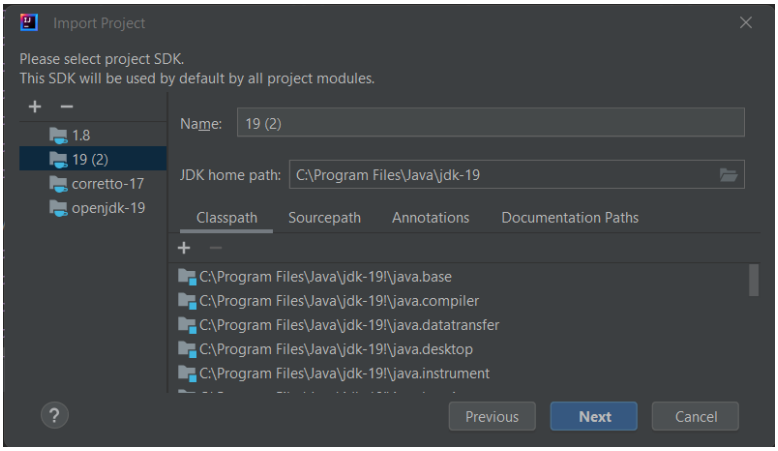
3. Tras ello, seguimos avanzando, seleccionando la opción de **"Create project from existing sources":**



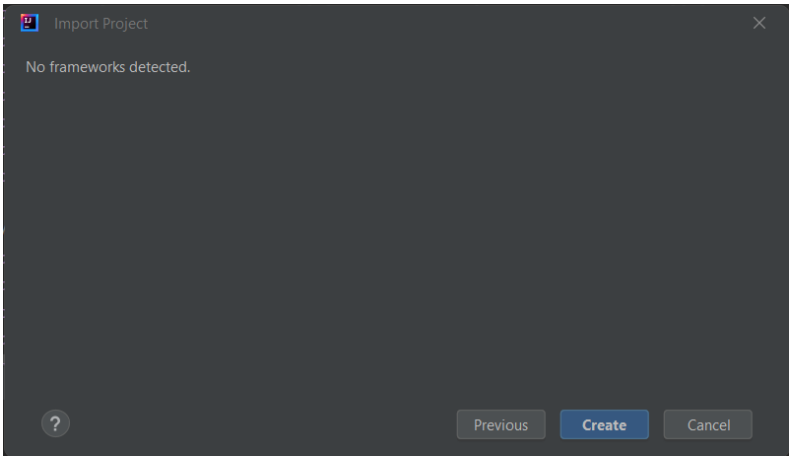
4. Seguimos hacia adelante, manteniendo el nombre de la **carpeta:**



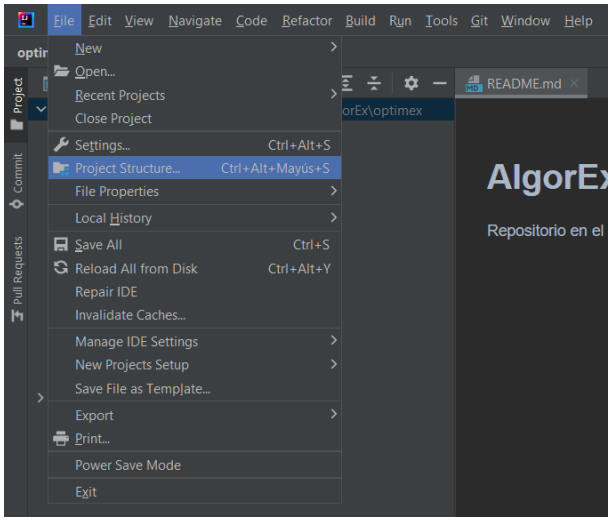
5. Seguimos avanzando y seleccionamos el **SDK** para el **proyecto**. Por lo general, es conveniente elegir la **versión más reciente** de **JDK** que tengamos:

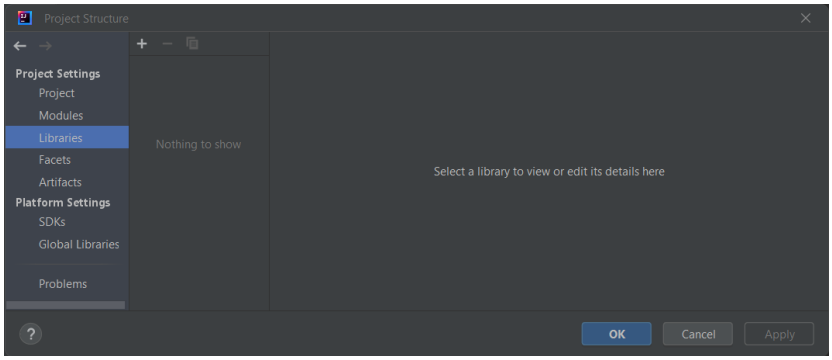


6. Tras ello, pulsamos sobre el botón de **Create** para crear el proyecto:

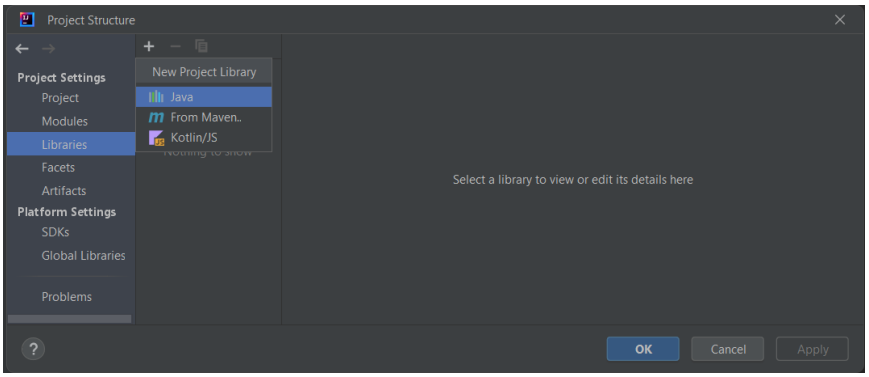


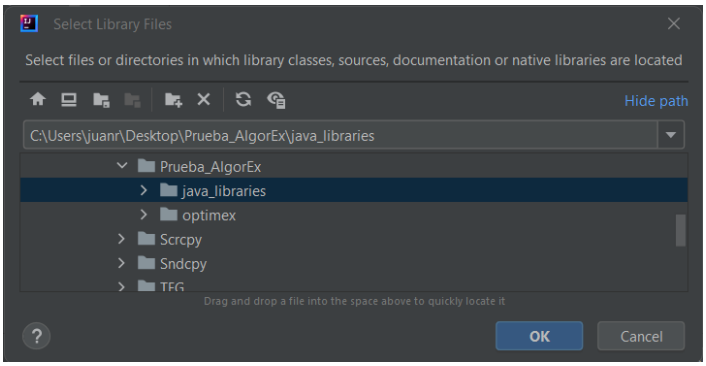
7. Una vez importado el **proyecto**, el siguiente paso es importar las librerías de la carpeta **java_libraries**. Para ello, pulsamos sobre **File → Project Structure…**

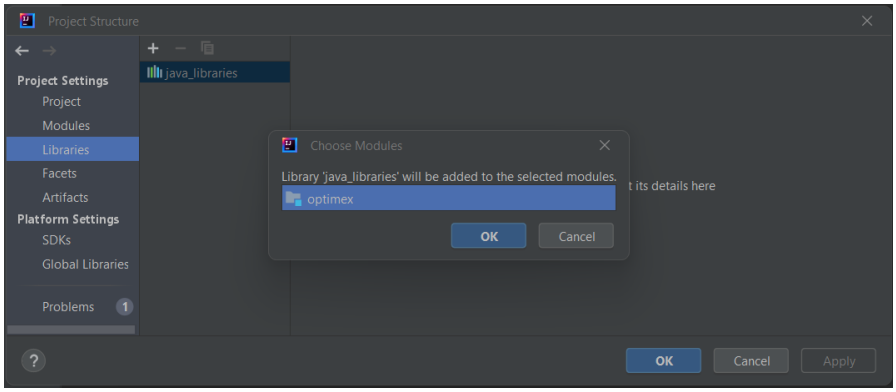


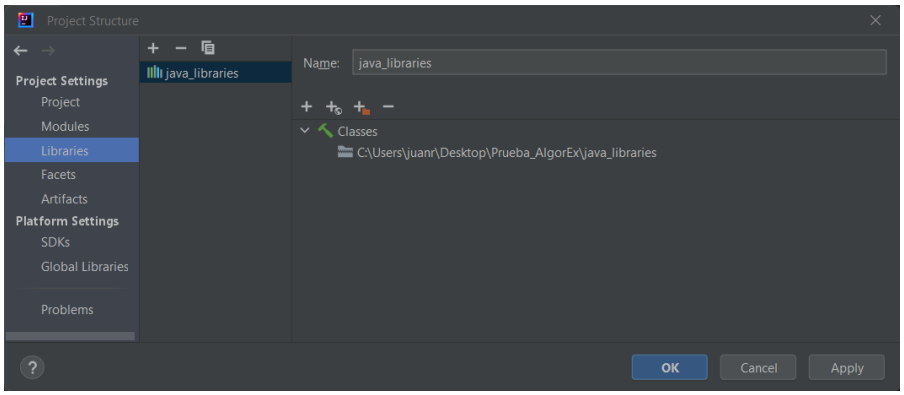


8. Dentro de la sección **Libraries**, pulsamos el botón del símbolo +, después la opción de **Java** y, en el **diálogo** que se genera, seleccionamos la carpeta **java_libraries**, y tras ello, aplicamos los cambios:

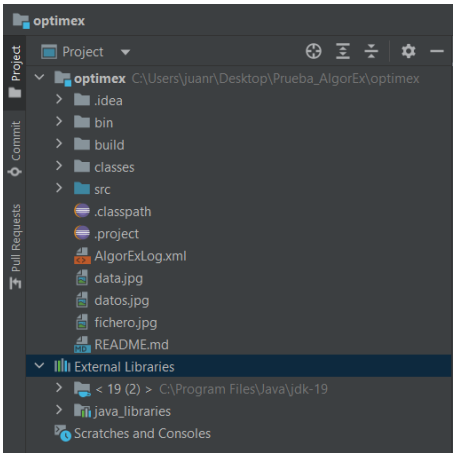






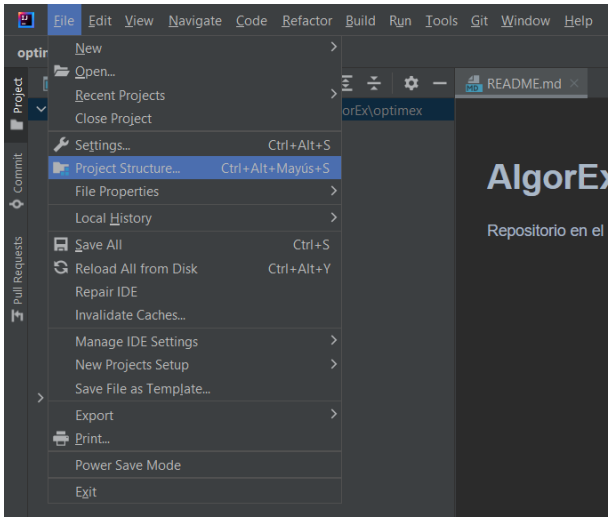


9. Una vez hecho eso, ya tendremos todo listo para ponernos a trabajar con el **sistema AlgorEx**.

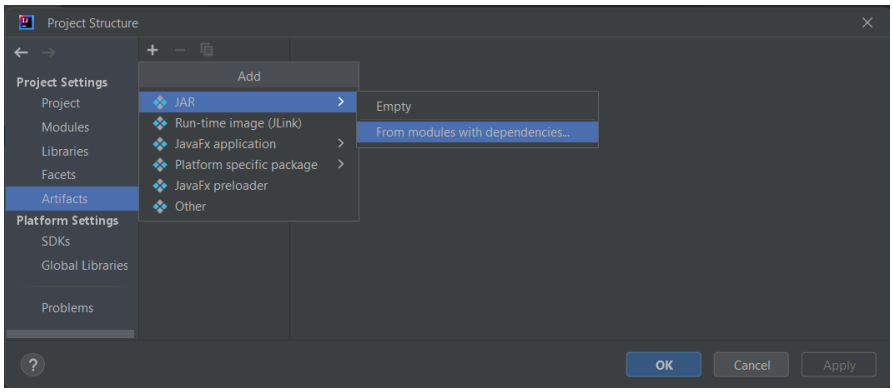


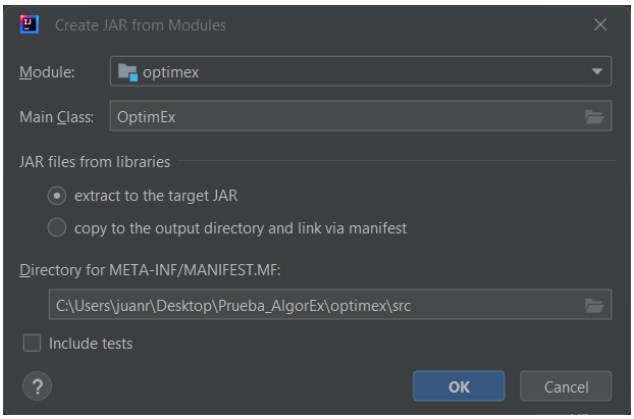
Si se desea generar un **fichero .jar** para poder tener el proyecto de forma más accesible, es necesario hacer lo siguiente:

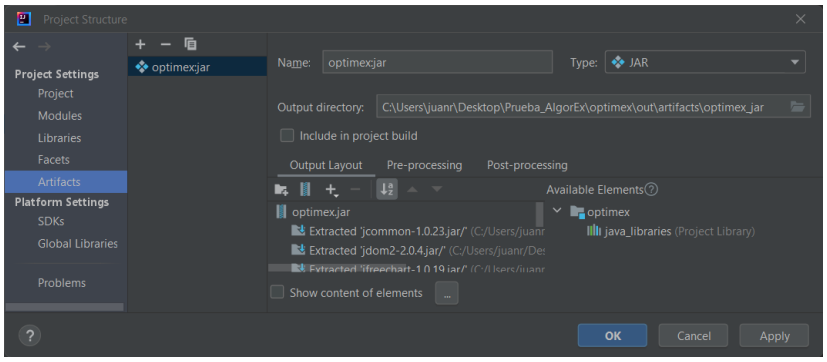
1. Pulsamos sobre **File → Project Structure…**



2. Dentro de la sección **Artifacts**, pulsamos el botón del símbolo **+**, después la opción de **JAR → From modules with dependencias…** y, en el **diálogo** que se genera, seleccionamos el module **optimex** y como clase principal **OptimEx**, y aplicamos los cambios:







3. Una vez hecho eso, cada vez que queramos generar el **fichero .jar**, pulsamos sobre **Build → Build Artifacts…**, y luego sobre el desplegable que aparece, pulsamos sobre **Build:**