Trabajo Práctico 3: Algoritmos golosos

# Encabezado

En este informe pretendemos explicar cómo se implementó la solución al trabajo práctico nº 3 de la materia. Este consiste en implementar un algoritmo goloso para resolver un problema sencillo de asignación de materias a una cantidad de aulas, dependiendo de su horario.

Las materias y sus horarios deberían ser cargadas en un archivo JSON especificando su hora de inicio y finalización. Dichas horas pueden ser números enteros. El programa debe calcular la mejor asignación de aulas que pueda y mostrarlo en una tabla.

# Idea

La idea es representar las materias como vértices en un grafo. Este grafo tendría aristas en aquellos vértices que representan materias cuyos horarios se superponen. Según la Teoría de Grafos, todo conjunto de vértices forma un *conjunto independiente* si y solo si no son adyacentes entre sí, es decir que no poseen aristas entre ellos. La técnica de *coloreo* permite identificar estos conjuntos “pintando” cada vértice con un color propio en caso de que no sean adyacentes entre sí.

De este modo, el algoritmo implementado le asigna un “color” a aquellos vértices que no poseen aristas entre sí, es decir, a aquellas materias que no se superponen en horario.

# Clases Principales

**Ventana:** Es la clase principal. Posee el *main* que hace correr la interfaz gráfica. También tiene un método para cargar un archivo desde el explorador y otro que devuelve el itinerario armado en una tabla.

**Tabla:** Hereda de JDialog, que permite mostrar elementos emergentes desde una ventana de Swing. Recibe un itinerario, posee métodos que arman las filas y columnas, y un método *mostrar* que hace que la ventana aparezca.

**Interfaz:** esta clase representa un controlador de las demás clases de negocio inferiores. Posee un método *obtenerItinerario* que recibe una ruta de archivo, y devuelve una instancia del itinerario con los datos cargados.

**Loader:** se vale de la librería GSON y permite parsear un archivo JSON desde una ruta.

**Itinerario:** representa un itinerario de materias y aulas. Recibe una lista de materias y un coloreo, y facilita un método para obtener el itinerario con un formato usable.

**GrafoMaterias:** hereda de Grafo y le agrega una lista de materias. Posee un método privado que agrega aristas al grafo entre los vértices que representan materias que se superponen. Luego tiene un método público para obtener el coloreo, que devuelve un diccionario en el que la clave es el ‘color’ y los valores una lista de vértices. Los colores se representan con números enteros.

# Aspectos de la implementación

1\_El hecho de que GSON permita importar archivos resulta útil a la hora de instanciar objetos. Nuestro **Loader** tiene como variable un *archivoJSON*  en la que se instancia el objeto con los datos del archivo de texto. De este modo no es necesario procesar la información. Sí es molesto y poco amigable estructurar el archivo para que responda a la forma de la clase.

2\_Debido a cuestiones de tiempo, la clase **Materia** no posee una implementación de horario vistosa, sino que simplemente se remite a recibir enteros. De hecho el programa funciona bien siempre y cuando la hora de inicio sea un número entero menor a la hora de finalización (en caso contrario lanza excepción), y es por eso que para mostrarlo usamos la expresión ‘1330’ para nombrar una y media de la tarde. Tenemos claro que en un software más complejo existen librerías que permiten formatear horas, y que cuando se trabaja con una base de datos y un framework, por lo general existe una API que provee métodos relacionados con el horario.

3\_El método que calcula la asignación de materias a aulas resulta bastante fácil de pensar, es goloso en el aspecto de que sólo se preocupa por su localidad, sin tener en cuenta otras posibilidades. De hecho, parte siempre desde el vértice cero, lo cual propone que pueden existir otras posibles distribuciones de aulas que son mejores.

# Conclusión

Teoría de Grafos resulta útil para resolver todo tipo de problemas, y los algoritmos están bastante bien definidos, lo cual permite diseñarlo de manera simple. En caso de los algoritmos golosos es sencillo, pero no lo es tanto con algoritmos más pesados como BackTracking. Para problemas que no son complejos de resolver o que no requieren una solución muy exacta, el diseñado en este trabajo funciona bien.

### Material consultado:

* Ana Laura Maffei. *Notas de Matemática Discreta*, UNGS.