TP Final Comisión 2.

Integrantes:

- Lucas Capocasa.
- Bruno Porfidio.
- Ariel Rey.

1. **Introducción:**

La gestión de datos médicos es un aspecto fundamental en el ámbito de la salud moderna, especialmente en un mundo donde la digitalización y el acceso a la información son cada vez más relevantes. Elegímos este tema porque consideramos que el manejo eficiente de datos médicos puede contribuir significativamente a la mejora de la atención al paciente y a la optimización de procesos en el sector salud. Además, la programación en Python es una habilidad valiosa que permite desarrollar soluciones innovadoras en este campo.

2. **Decisiones de diseño:**

Iniciaremos en **main.py,** donde decidimos crear un menu que despliegue las opciones disponibles en el programa. Utilizamos un ciclo **while** con la variable booleana **sigue_eligiendo = True** para asi de esta forma, siempre y cuando el usuario decida seguir utilizando el programa, seguiremos dentro del ciclo. En caso de tocar la opción 21 (**Salir**), la variable **sigue_eligiendo** pasara a ser falsa, por lo cual el programa se cerrará.
Dentro de lo que son las opciones del menú, se tomo la decisión de que cada opción tenga la menor cantidad de lineas de código posibles, siendo idealmente solo una linea que llame a una función en particular según se requiera el caso. En el único de estos casos que no utilizamos pocas lineas de código fue en las opciones 16 y 17 (**Búsqueda BFS y DFS**), pero si reciclamos bastante código con la función **obtener_id_hospital()**, repitiéndola en cada opción, pero con un mensaje distinto par adaptar su funcionalidad a cada caso particular.

Dentro de **ArbolAVL.py** tenemos la clase **ArbolAVL**, junto con su **NodoAVL.** Un Arbol AVL seria un árbol balanceado, por lo cual lo elegimos para poder mejorar la búsqueda de pacientes dentro del árbol mediante su id y que no quede un árbol desbalancedo, en el cual si quiero llegar al ultimo paciente tenga que recorrer todo el árbol completo en vez de evaluar sus nodos izquierdo y derecho.

En la clase **Paciente**, dentro de **Paciente.py,** creamos un paciente que tiene como atributos un ID, nombre, edad, gravedad, un id del hospital en el que se encuentra, un historial clínico y un listado de medicamentos y enfermedades. Dentro de esta clase, destacamos la diferenciación entre el método **__str_()** y el método **detalle_completo(),** el cual decidimos crear ya que si nosotros utilizábamos siempre el método **__str__ ()** para mostrar los datos del paciente, si este tuviese un historial clínico muy amplio, se sobrecargaría de información la pantalla. Esta decisión es fácilmente revertible si se desea, solo habría que copiar y pegar el

contenido de **detalle_completo()** dentro del método __**str**_ **()**, y luego realizar las modificaciones estéticas que se deseen.

Luego tenemos las clases **Medicamento** y **Enfermedad**, las cuales son idénticas ya que ambas poseen un nombre y una fecha, y a su vez poseen el método __**str**__ **()** para poder mostrar la información de cada una por pantalla cuando se necesite. Similar a ellas tenemos **EventoMedico,** clase la cual tiene como atributos un tipo (consulta, diagnóstico o tratamiento), detalles del tipo, y una fecha que por defecto es la fecha y hora la cual se agrega la consulta.

En **Hospital.py** tenemos la clase **HOSPITAL,** la cual posee un ID de hospital para diferenciarlo, un nombre, una especialidad y sus conexiones con otros hospitales, ademas del método __**str**__ **()** para mostrar sus datos por pantalla. En este punto tomamos la decisión de que cada hospital tenga una sola especialidad para ahorrar en memoria, ya que si nosotros tenemos que buscar una especialidad puntual en muchos hospitales que tengan muchas especialidades, la búsqueda nos llevaría a hacer un gasto de memoria muy alto, dependiendo de la cantidad de especialidades que tenga cada hospital.

En **GrafoHospital.py** tenemos la clase **GrafoHospitales**, la cual tiene todas las funcionalidades de los hospitales que se utilizan en el menú, como pueden ser **agregar_hospital(), agregar_conexion(), mostrar_hospitales()**, etc. Aquí se destacan las funciones **hospitales_ejemplos()** y **conexiones_ejemplos()**, las cuales creamos para que el usuario que desee probar la funcionalidad del programa para detectar errores, no tenga que crear hospitales y conexiones cada vez. En caso de querer utilizar el sistema desde 0, simplemente se eliminan o comentan estas funciones en **main.py** y **GrafoHospital.py.**

Por ultimo, la parte mas importante de nuestro código, el **GestorPacientes** dentro de **Gestor.py.** Esta clase es la mas importante, ya que es el paso siguiente a todas las opciones que seleccionamos dentro de **main**. Esta clase es quien gestionara todas las opciones que el usuario elija, y lo irá dirigiendo dentro de las demás partes del código. Como se destacaron en el párrafo anterior las funciones **hospitales_ejemplos()** y **conexiones_ejemplos()**, aquí se destaca **pacientes_ejemplos()**, la cual como su nombre indica, crea pacientes ficticios para poder probar el código. En esta parte del código se destacan una gran cantidad de validaciones, es por eso que tiene tantas lineas, pero son estrictamente necesarias para lograr la robustez del programa.