# Parcial 4

Introducción:

**Breve descripción del problema seleccionado: Árbol genealógico**

El problema seleccionado trata de ilustrar un árbol genealógico el cual es una representación gráfica de como esta relacionados diferentes miembros de una familia a través de una estructura jerárquica como lo son los árboles, de esta manera no solo tenemos acceso a todos ancestros o sucesores de una persona sino también poder buscar relaciones entre dos miembros.

**Explicación de la relevancia del problema en aplicaciones prácticas**

Este proyecto sobre arboles genealógicos es muy útil en nuestro alrededor debido a que puede tener diversos usos por ejemplo dentro de una familia si nos enfocamos en el tema de la salud pueden ser muy útiles para determinar variantes genéticas patogénicas, rasgos y enfermedades, fuera de que también son muy útiles para determinar parentescos y además para indicar que familiares ya fallecieron además que ayuda a que cualquier persona pueda saber exactamente de dónde viene.

Fundamentos Teóricos:

**Descripción detallada de los conceptos teóricos relacionados con el ejercicio**

Un árbol es una estructura jerárquica que consiste en nodos conectados mediante aristas o ramas, un árbol consta de un nodo raíz, que es el nodo principal del árbol en nuestro caso seria el mayor ancestro, este ancestro puede tener cero o más subárboles y cada nodo tiene un nodo padre excepto el nodo raíz que seria el nodo más anciano y aquellos nodos que serian los miembros de la familia que no tienen decendencia son los nodos hoja y los nodos padre de un nodo son los antecesores de ese nodo.

Links :

<https://www.lawebdelprogramador.com/foros/Python/2188686-Arbol-genealogico-en-python-con-recursividad.html>

<https://www.youtube.com/watch?v=_6Fmuw-lGNk>

<https://www.lucidchart.com/blog/es/como-hacer-un-arbol-genealogico>

<https://definicion.de/arbol-genealogico/>

Desarrollo del Código:

**Explicación de la lógica y los pasos seguidos para la implementación del código.**

**clase Nodo:**

Se define una clase nodo para representar a cada individuo en el árbol genealógico.

Cada nodo tiene un nombre, un padre, una madre y una lista de hijos.

**clase ArbolGenealogico:**

Se define una clase ArbolGenealogico para representar el árbol

La clase contiene un diccionario miembros para almacenar todos los nodos del árbol, organizados por su nombre.

También contiene un diccionario niveles para almacenar el nivel de cada nodo en el árbol.

**Método agregar\_miembro:**

Este método permite agregar un nuevo miembro al árbol genealógico. Toma el nombre y la edad del miembro como entrada y crea un nuevo nodo con esa información.

**Método agregar\_relacion\_padre\_hijo:**

Este método establece la relación padre-hijo entre dos miembros del árbol genealógico.

Verifica si ambos nombres están presentes en el árbol.

Asigna al hijo el nodo correspondiente como padre y agrega al hijo a la lista de hijos del padre.

Llama al método asignar\_niveles después de establecer la relación para mantener actualizados los niveles.

**Método asignar\_niveles:**

Este método asigna niveles a todos los nodos del Arbol genealógico.

Utiliza un recorrido recursivo para asignar niveles a cada nodo, comenzando desde las raíces y avanzando hacia abajo.

Para identificar las raíces, busca nodos que no tengan padre ni madre.

Llama a la función interna asignar\_niveles\_recursivo para asignar niveles a cada nodo.

**Método imprimir\_arbol:**

Este método imprime el árbol genealógico para el usuario.

Itera sobre todos los miembros del árbol y muestra el nombre, los padres, los hijos, los hermanos y los nietos de cada miembro.

**Método determinar\_relacion:**

Este método determina la relación entre dos miembros del árbol genealógico.

Mira la diferencia de niveles entre los dos miembros y utiliza esta resta para establecer el parentesco.

si la diferencia es 0, son hermanos

si es 1, son padres/hijos

si es 2, son abuelos/nietos;

sino la relación es más lejana.

**Método graficar\_arbol:**

Este método utiliza la biblioteca NetworkX y Matplotlib para visualizar el árbol genealógico como un gráfico.

Crea un “árbol” donde cada nodo representa un miembro y cada arista

representa la relación padre-hijo.

Utiliza la función graphviz\_layout para obtener una disposición jerárquica y luego dibuja el “árbol” utilizando Matplotlib.

**Método agregar\_miembro\_con\_relaciones:**

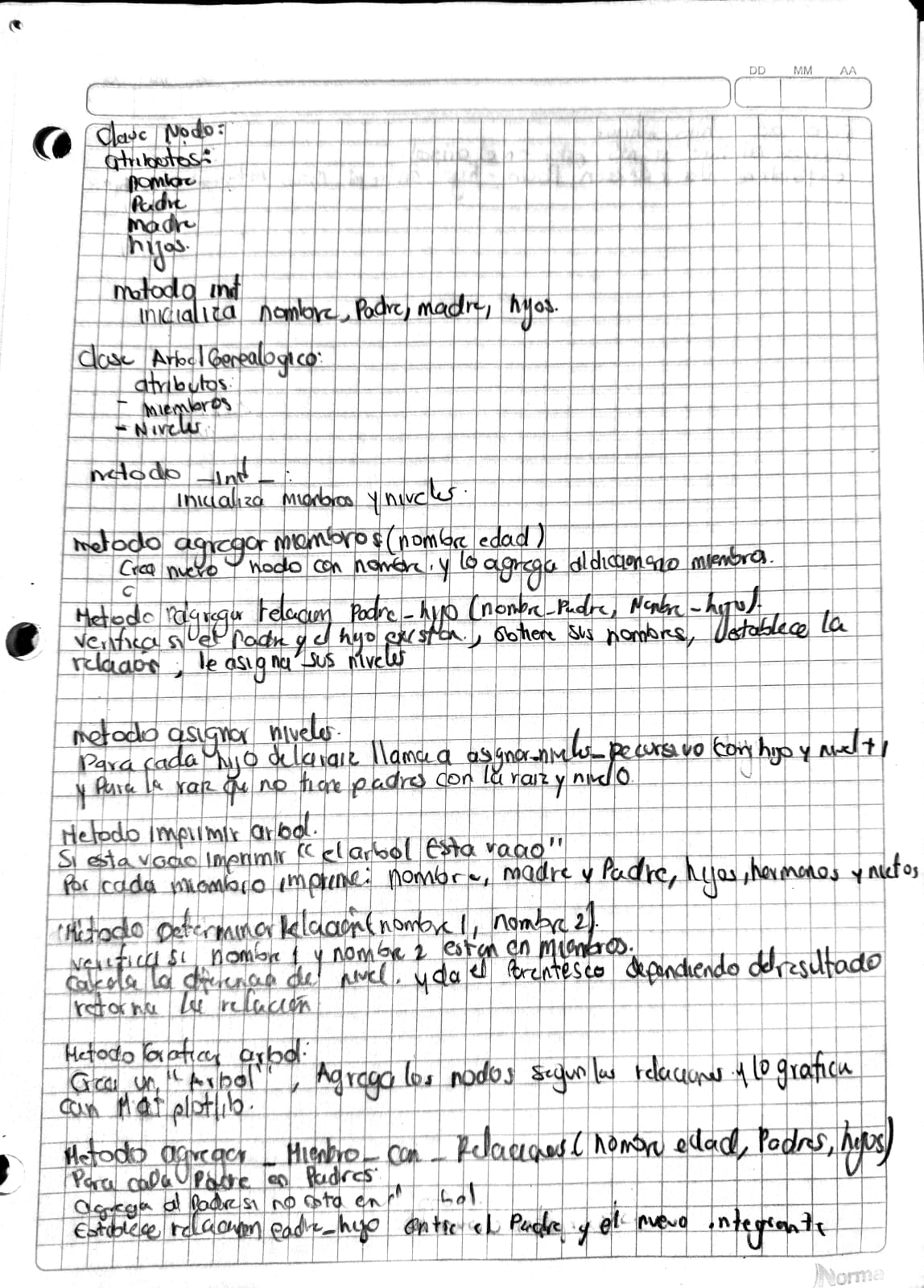
Este método agrega un nuevo miembro al árbol genealógico y establece relaciones padre-hijo con los padres y los hijos proporcionados.

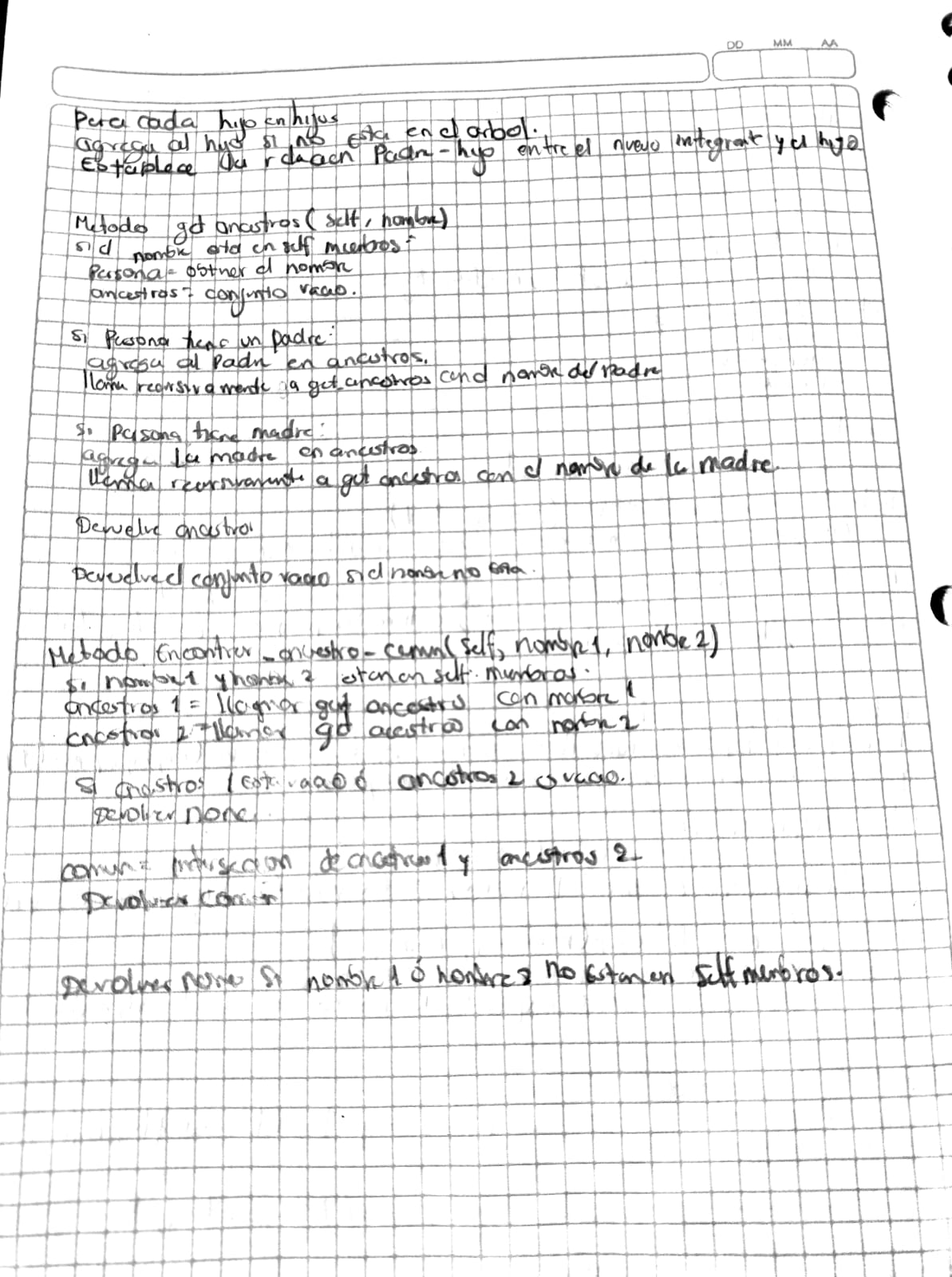
Llama a los métodos agregar\_miembro y agregar\_relacion\_padre\_hijo según sea necesario.

**Función menu:**

Esta función proporciona un menú para que el usuario pueda interactuar con el árbol genealógico. tiene de opciones agregar miembros con relaciones, imprimir el árbol, graficar el árbol, determinar el parentesco entre dos miembros o salir del programa.

**Pseudocódigo**





Clase Nodo:

**Propiedades:**

- nombre

- padre

- madre

- hijos

**Método \_\_init\_\_:**

Inicializar nombre, padre, madre, hijos

**Clase ArbolGenealogico:**

Propiedades:

- miembros

- niveles

**Método \_\_init\_\_:**

Inicializar miembros y niveles

**Método agregar\_miembro(nombre, edad):**

Crear un nuevo nodo con nombre y edad

Agregar el nodo al diccionario miembros

**Método agregar\_relacion\_padre\_hijo(nombre\_padre, nombre\_hijo):**

Verificar si nombre\_padre y nombre\_hijo están en miembros

Obtener los nodos correspondientes a nombre\_padre y nombre\_hijo

Establecer la relación padre-hijo entre los nodos

Llamar al método asignar\_niveles

**Método asignar\_niveles:**

Función interna asignar\_niveles\_recursivo(raiz, nivel):

Si raiz es None, retornar

Asignar nivel a raiz en niveles

Para cada hijo de raiz:

Llamar recursivamente a asignar\_niveles\_recursivo con hijo y nivel + 1

Encontrar las raíces del árbol (nodos sin padres ni madres)

Para cada raíz:

Llamar a asignar\_niveles\_recursivo con la raíz y nivel 0

**Método imprimir\_arbol:**

Si miembros está vacío, imprimir "El árbol está vacío"

Para cada nombre y nodo en miembros:

Imprimir nombre

Imprimir padres y madres del nodo

Imprimir hijos del nodo

Imprimir hermanos del nodo

Imprimir nietos del nodo

**Método determinar\_relacion(nombre1, nombre2):**

Verificar si nombre1 y nombre2 están en miembros

Calcular la diferencia de niveles entre nombre1 y nombre2

Determinar la relación basada en la diferencia de niveles

Retornar la relación

**Método graficar\_arbol:**

Crear un grafo dirigido

Agregar nodos y bordes al grafo según las relaciones padre-hijo

Graficar el grafo utilizando Matplotlib

**Método agregar\_miembro\_con\_relaciones(nombre, edad, padres, hijos):**

Agregar un nuevo miembro con nombre y edad

Para cada padre en padres:

Agregar el padre al árbol si no está presente

Establecer la relación padre-hijo entre el padre y el nuevo miembro

Para cada hijo en hijos:

Agregar el hijo al árbol si no está presente

Establecer la relación padre-hijo entre el nuevo miembro y el hijo

**Método get\_ancestros(self, nombre):**

Si nombre está en self.miembros:

persona = Obtener el miembro con el nombre nombre desde self.miembros

ancestros = Conjunto vacío

Si persona tiene un padre:

Agregar el nombre del padre de persona al conjunto ancestros

Llamar recursivamente a get\_ancestros con el nombre del padre de persona y agregar los resultados al conjunto ancestros

Si persona tiene una madre:

Agregar el nombre de la madre de persona al conjunto ancestros

Llamar recursivamente a get\_ancestros con el nombre de la madre de persona y agregar los resultados al conjunto ancestros

Devolver ancestros

Devolver conjunto vacío si nombre no está en self.miembros

**Método encontrar\_ancestro\_comun(self, nombre1, nombre2):**

Si nombre1 está en self.miembros y nombre2 está en self.miembros:

ancestros1 = Llamar a get\_ancestros con nombre1

ancestros2 = Llamar a get\_ancestros con nombre2

Si ancestros1 es vacío o ancestros2 es vacío:

Devolver None

comun = Intersección de ancestros1 y ancestros2

Si comun no está vacío:

Devolver comun

Devolver None si nombre1 o nombre2 no están en self.miembros

**Función menu:**

Crear un árbol genealógico

Agregar algunos miembros y establecer relaciones padre-hijo

Mientras verdadero:

Imprimir opciones de menú

Leer opción seleccionada

Si la opción es 1:

Leer nombre, edad, padres y hijos del nuevo miembro

Llamar a agregar\_miembro\_con\_relaciones con los datos proporcionados

Si la opción es 2:

Llamar a imprimir\_arbol

Si la opción es 3:

Llamar a graficar\_arbol

Si la opción es 4:

Leer nombres de dos miembros

Llamar a determinar\_relacion con los nombres proporcionados

Si la opción es 5:

Salir del bucle

Resultados:

Casos base:

familia.agregar\_miembro("Roberto", 70)

familia.agregar\_miembro("Juan", 50)

familia.agregar\_miembro("Carlos", 30)

familia.agregar\_miembro("Pedro", 25)

familia.agregar\_miembro("Maria", 20)

familia.agregar\_miembro("Luisa", 18)

# Establezco relaciones padre-hijo

familia.agregar\_relacion\_padre\_hijo("Roberto", "Juan")

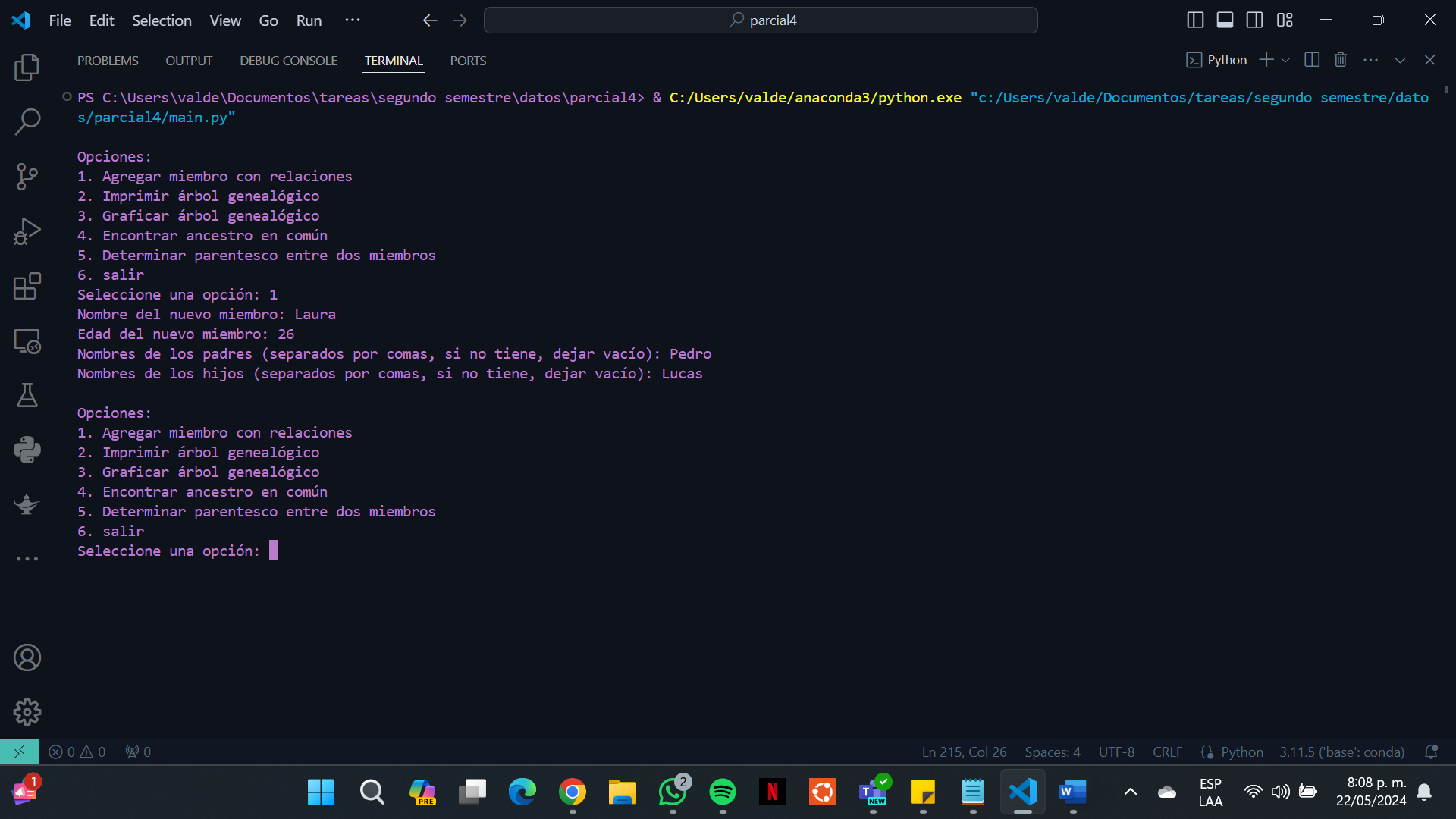
familia.agregar\_relacion\_padre\_hijo("Juan", "Carlos")

familia.agregar\_relacion\_padre\_hijo("Juan", "Pedro")

familia.agregar\_relacion\_padre\_hijo("Juan", "Maria")

familia.agregar\_relacion\_padre\_hijo("Juan", "Luisa")

**menú opcion 1**



**Menú opcion 2**

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

**Menú opcion 3**

Captura de pantalla de computadora

Descripción generada automáticamente

**Menú opcion 4**

Captura de pantalla de computadora

Descripción generada automáticamente

**Menú opcion 5**

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

**Análisis de los resultados obtenidos**

Este código arroja los resultados esperados en las pruebas, además de que realiza correctamente las tareas requeridas de poder agregar miembros, decirme los parentescos entre dos nodos, y determinar el ancestro común más reciente entre dos nodos todo ejecutándose de manera correcta.

***Reflexión sobre la solución implementada y su eficiencia***

La solución implementada y se ejecuta correctamente, aunque en cuanto a su eficiencia no estoy segura de que sea la mejor debido a que se podrían implementar mejoras para la determinación de relaciones para así mejorar su rendimiento

**Posibles mejoras o extensiones del trabajo realizado.**

Entre las posibles mejoras que me gustaría hacerle a mi código esta que llegue a ser muy eficiente con un gran volumen de datos para el árbol o por ejemplo la lectura desde un archivo que contenga las relaciones