Trabajo Práctico - Semana de integración II

Conjuntos

Alumnos

Santiago Octavio Varela (santiagov.linked@gmail.com) ,

Ximena Maribel Sosa (ximenasosa44@gmail.com)

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Matemática

Docente Titular

Martina Wallace

Docente Tutor

Fernanda Espósito

Fecha de entrega

13 de Junio de 2025

Parte 1 – Desarrollo Matemático (Conjuntos y Lógica)

1- Dado que está la posibilidad de generar números de DNI ficticios en la parte 2 de este trabajo, decidimos crearlos directamente también para este punto mediante el sitio https://www.random.org/integers/ . Estos fueron los resultados:

- DNI Nº1 = 26.931.802
- DNI Nº2 = 28.389.594

2- Conjuntos en base a ambos DNI, en el mismo orden, colocar más de una vez los elementos que se repitan

Conjunto A (en base a DNI º1) = {2, 6, 9, 3, 1, 8, 0}

Conjunto B (en base a DNI º2) = {2, 8, 3, 9, 5, 4}

3-

a) Unión:

$A \cup B = \{0, 1, 2, 3, 4, 5, 6, 8, 9\}$

b) Intersección:

$A \cap B = \{2, 3, 8, 9\}$

c) Diferencia entre pares:

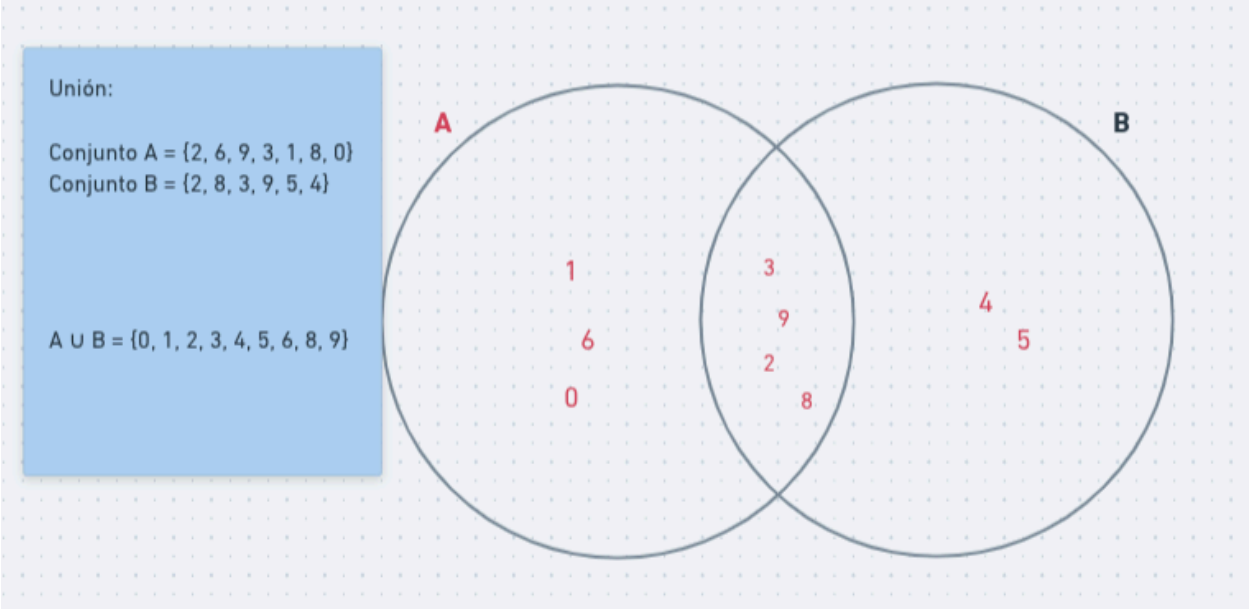
$A - B = \{0, 1, 6\}$
$B - A = \{5, 4\}$

d) Diferencia simetrica

$A \triangle B = \{0, 1, 4, 5, 6\}$

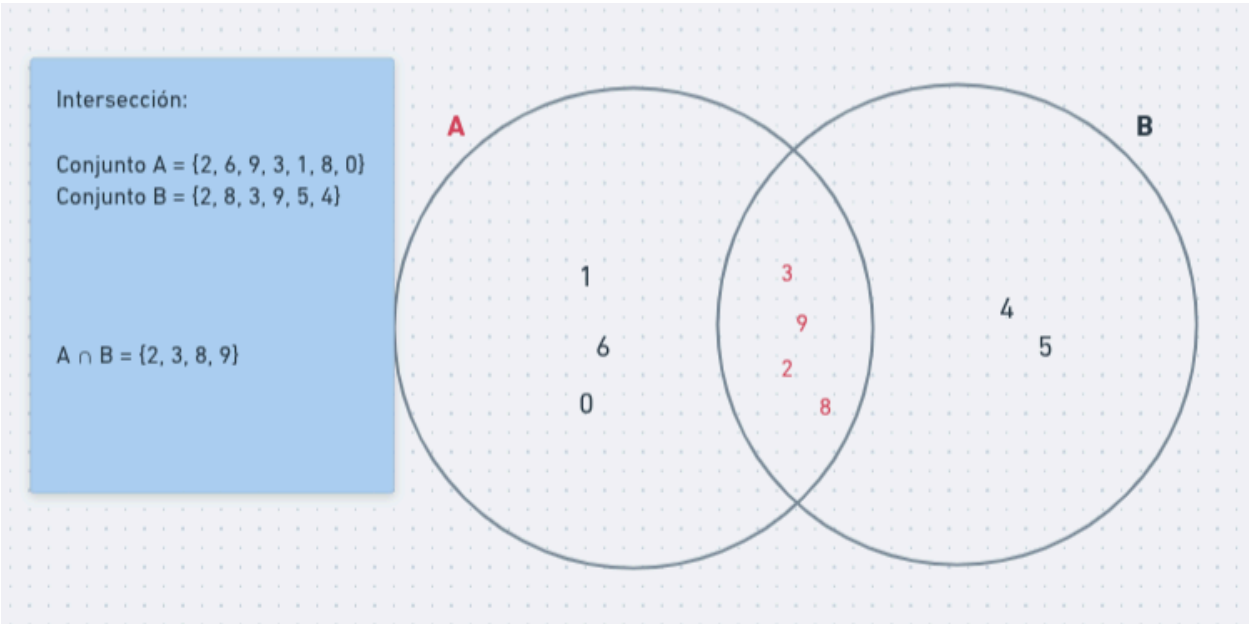
4-

Diagrama de Venn para Unión:



Cabe aclarar que, dado que estamos hablando de una unión, todos los números están en color rojo justamente para dar cuenta de la unión.

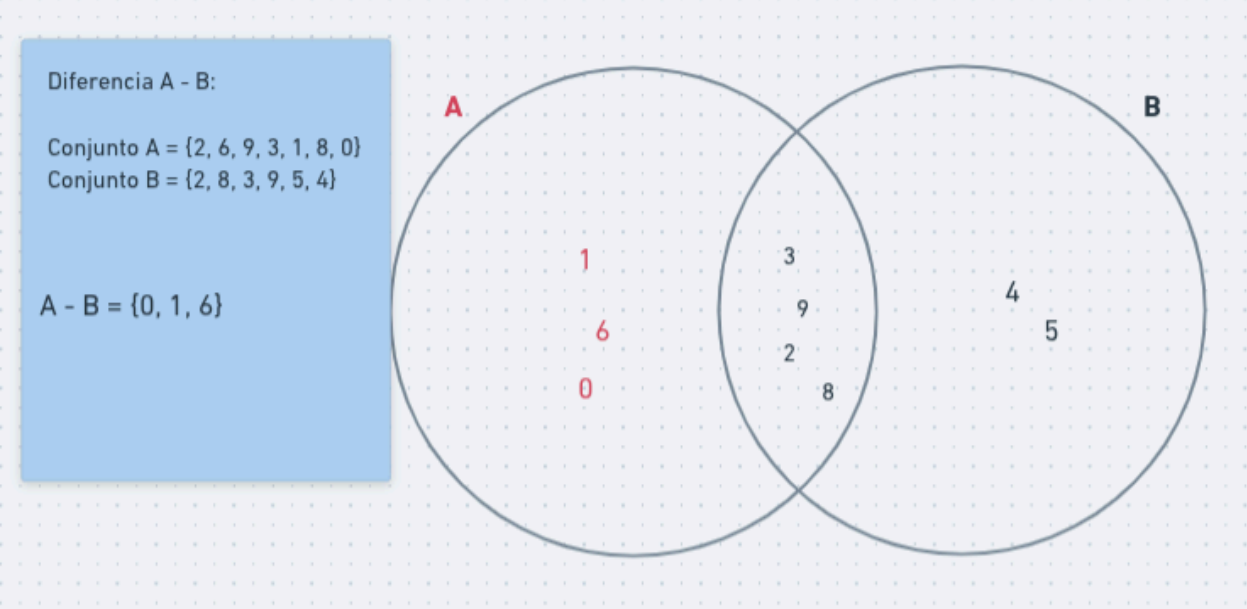
Diagrama de Venn para Intersección:



Ahora, los elementos que están en rojo son aquellos que son parte de la intersección entre A y B.

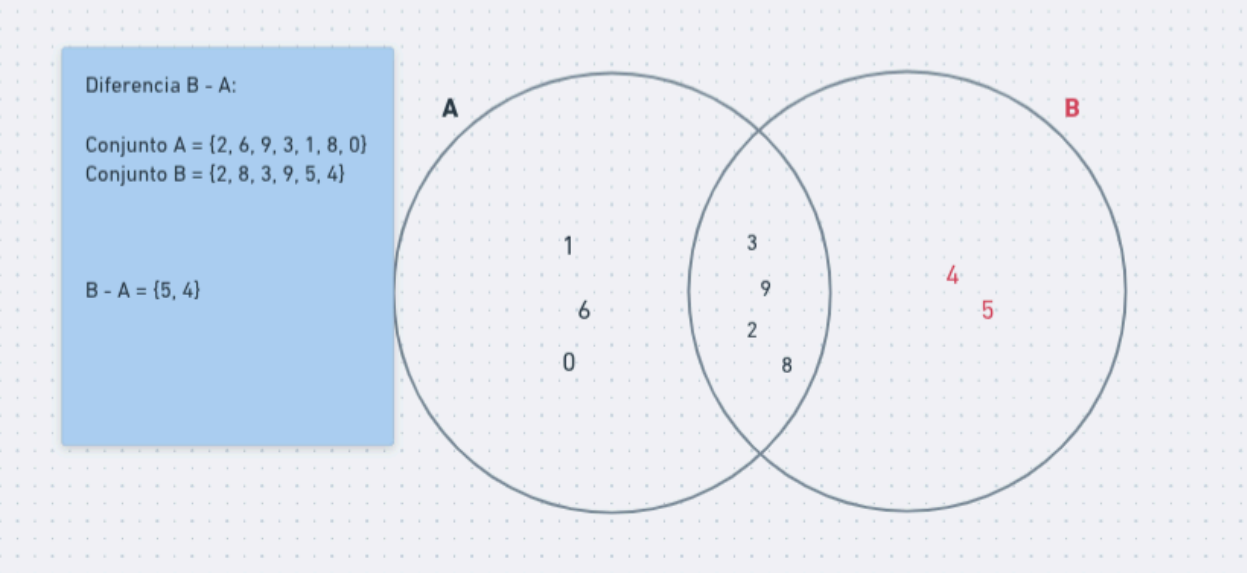
Diagrama de Venn para Diferencia entre pares:

a) A - B = {0, 1, 6}



En este diagrama, en rojo están todos los números que solo pertenecen a A, para dar cuenta de la diferencia entre A - B.

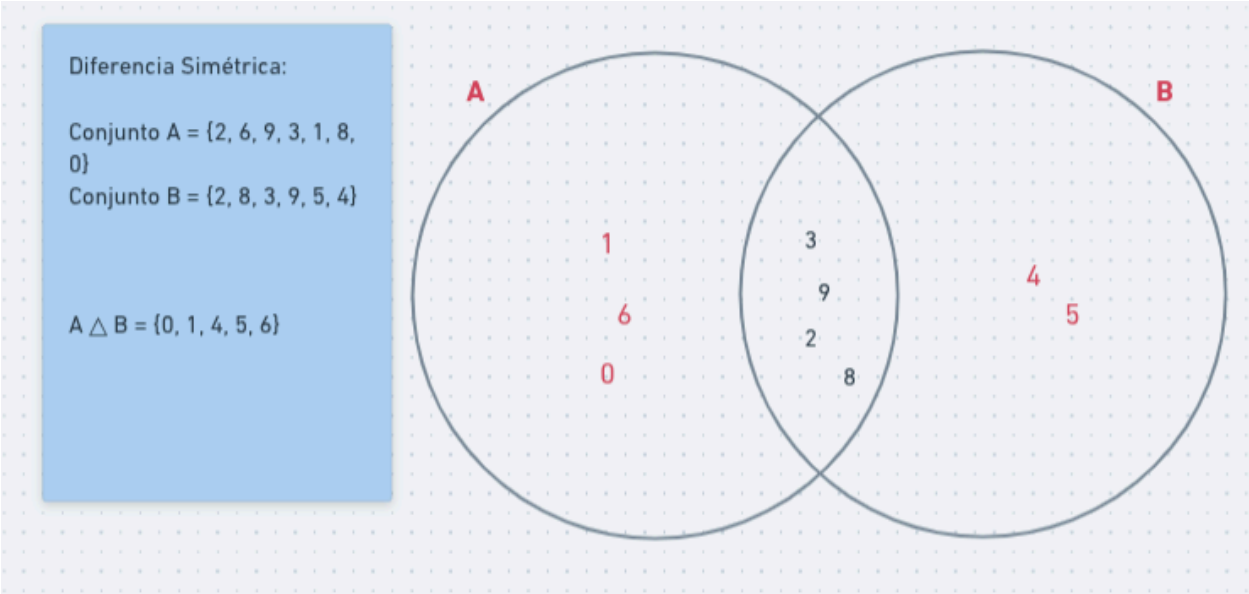
b) B - A = {5, 4}



En este diagrama se invierte la importancia respecto al anterior, ya que en rojo están todos los números que solo pertenecen a B, para dar cuenta de la diferencia entre B - A.

Diagrama de Venn para Diferencia Simétrica:

$A \triangle B = \{0, 1, 4, 5, 6\}$



Relacionado con los diagramas de diferencia anteriores, la diferencia simétrica básicamente contiene en el diagrama ambas diferencias. Por lo tanto, en rojo están los elementos de la diferencia resultante de A - B = {0, 1, 6} y las de B - A = {4, 5}, mientras que en negro están los elementos correspondientes a la intersección entre A y B. Esta operación se escribe como $A \triangle B = (A - B) \cup (B - A)$.

5. Algunas expresiones lógicas que podemos considerar, en base a los ejercicios previos, son las siguientes:

a) Expresión planteada pensando en el rango numérico compartido y las intersecciones:

   "Si los conjuntos A y B comparten más de 3 números y cuando los junto tengo por lo menos 8 números distintos, entonces hay una relación equilibrada entre ellos."

b) Expresión con una lógica similar a la anterior, pero distinto planteamiento:

   "Si al juntar A y B tenemos todos los números del 0 al 9 y, además, comparten al menos 4 números, entonces cubren todo el rango numérico."

c) Expresión planteada teniendo en cuenta cantidad de pares e impares en cada conjunto:

   "Si más de la mitad de todos los números (juntando A y B) son pares, pero hay al menos un número impar que solo está en uno de los dos conjuntos, entonces los pares dominan salvando esa excepción."

Parte 2 – Desarrollo del Programa en Python y video explicativo

Todas las URL que aparecerán también estarán en el Anexo, junto con el código completo.

Evaluación de condiciones / expresiones lógicas

Expresión usada para el programa: “Si los conjuntos A y B comparten más de 3 números y cuando los junto tengo por lo menos 8 números distintos, entonces hay una relación equilibrada entre ellos.”

Fundamentación de la expresión utilizada:

La expresión lógica se fundamenta en operaciones clásicas de teoría de conjuntos. Esta condición se implementa dentro de la función analizar_dnis(), donde se comparan los conjuntos de dígitos únicos de cada DNI. Por un lado, conjunto_a & conjunto_b evalúa la intersección, es decir, los dígitos que comparten ambos DNIs. Por otro, conjunto_a | conjunto_b calcula la unión, o sea, el total de dígitos distintos entre ambos.

La lógica combinada (intersección > 3 y unión ≥ 8) permite establecer un criterio para determinar si existe una coincidencia significativa entre los DNIs, sin que sean idénticos. El concepto de “relación equilibrada” surge de esta idea de equilibrio ya que hay coincidencias, pero también existe diversidad. Este tipo de razonamiento conecta la lógica matemática con un análisis más cualitativo. En definitiva, es un ejemplo concreto de cómo se puede aplicar una expresión condicional compuesta dentro de un programa estructurado.

Caso práctico: nuestro programa

URL del código completo en GitHub:
https://github.com/santiagovOK/matematica_integracion-dos_VarelaSosa/blob/main/matematica_integracion-dos_VarelaSosa.py

URL Repositorio en GitHub:
https://github.com/santiagovOK/matematica_integracion-dos_VarelaSosa/tree/main

Parte 3 – Video de Presentación en YouTube: https://youtu.be/uUo-6G-x5v0

Breve fundamentación de métodos utilizados y otros detalles

Para el desarrollo del programa que hace a este trabajo, recurrimos a métodos en Python que permiten aplicar operaciones matemáticas fundamentales dentro de estructuras de programación. Creemos que es pertinente mencionar esto, ya que investigamos sobre ellos en

la documentación de Python (Python Software Foundation, *n.d.*), específicamente para este trabajo.

Uno de los ejes centrales, por supuesto, fue el uso de conjuntos en Python. En particular, utilizamos los métodos .union() y .intersection() para realizar operaciones de teoría de conjuntos, visibles en la parte I de este trabajo, sobre los dígitos únicos de los DNIs ingresados. La función .union() permite obtener todos los elementos distintos que aparecen al menos una vez entre los conjuntos comparados, mientras que .intersection() devuelve solamente aquellos elementos que aparecen en todos ellos. Estas herramientas resultan esenciales para establecer relaciones lógicas entre los datos y construir criterios como la “relación equilibrada”, basada en la cantidad de dígitos compartidos y distintos.

Además, reutilizamos una función desarrollada en Programación I para el trabajo integrador: bubble_sort(). Este algoritmo de ordenamiento compara elementos adyacentes dentro de una lista y los intercambia si están en el orden incorrecto, repitiendo el proceso hasta que toda la lista esté ordenada. Aunque no es el método más eficiente para grandes volúmenes de datos, su lógica sencilla y explícita lo hace ideal para prácticas educativas y para aplicar con pequeñas listas, como las que se generan en nuestro caso.

Estos métodos fueron elegidos no solo por su utilidad funcional, sino también porque permiten visualizar cómo conceptos matemáticos como operaciones de conjuntos y ordenamientos se traducen en estructuras algorítmicas. Esto refuerza el vínculo entre el pensamiento lógico-matemático y la construcción de soluciones programadas.

Notas Adicionales

Reparto de tareas y trabajo colaborativo

El desarrollo del programa fue realizado de forma colaborativa y equitativa por ambos integrantes del grupo, combinando tareas individuales con espacios de integración y revisión mutua.

Santiago se encargó principalmente de la primera parte del programa, relacionada con el procesamiento de los DNIs. Desarrolló y adaptó funciones como ingresar_dnis(), contar_frecuencia_digitos(), obtener_digitos_unicos() y analizar_dnis(), donde se aplican operaciones de conjuntos como .union, .intersection y diferencia simétrica. También incorporó la función bubble_sort(), reutilizada del trabajo integrador de Programación I, para ordenar los resultados. Además, propuso la lógica de la “relación equilibrada” entre pares de DNIs en base a condiciones matemáticas.

Ximena trabajó principalmente en la segunda parte del programa, centrada en el análisis de años de nacimiento. Implementó las funciones es_bisiesto(anio) y analizar_anios(), que evalúan condiciones lógicas (como si los años son posteriores a 2000 o si hay años bisiestos), calculan edades y generan el producto cartesiano entre años y edades. También se encargó de validar el ingreso de datos y dar formato a la salida en consola.

En cuanto al desarrollo del análisis manual de los conjuntos, ambos integrantes participaron equitativamente. Ximena elaboró los conjuntos A y B con los dígitos correspondientes, y realizó las operaciones de unión, intersección, diferencia entre pares y diferencia simétrica. Santiago se ocupó de representar cada operación con diagramas de Venn y de redactar las explicaciones visuales asociadas. También formuló tres expresiones lógicas originales basadas en los conjuntos, abordando relaciones numéricas, cobertura total y distribución par/impar. Esta división permitió que cada parte fuera abordada con profundidad desde lo lógico, lo visual y lo matemático.

Por último, en cuanto al video, Ximena se encargó de elaborar un guión general para establecer un orden de presentación del caso práctico, mientras que Santiago se encargó de la edición del video.

Ambos participaron en la definición del enfoque general del programa, revisaron mutuamente el código escrito por el otro y realizaron pruebas conjuntas para asegurar el correcto funcionamiento del sistema completo. El trabajo fue dividido en partes, pero articulado en función del objetivo principal de este trabajo, que es integrar herramientas de matemática y programación de manera coherente y funcional en base al tema conjuntos.

Conclusiones finales

A lo largo de este trabajo integrador tuvimos la oportunidad de aplicar de forma concreta los conocimientos adquiridos sobre conjuntos, relaciones y operaciones lógicas, integrándolos con

programación en Python. Pudimos ver cómo conceptos teóricos, muchas veces abstractos, pueden modelar problemas reales y resolverse mediante el código.

Tomando como base dos DNIs, desarrollamos un programa que permite analizar los conjuntos de dígitos involucrados, aplicando operaciones como unión, intersección, diferencias y producto cartesiano. Además, integramos funciones condicionales, ordenamientos personalizados (como bubble sort) y una lógica de evaluación para determinar relaciones equilibradas entre los conjuntos.

Complementamos el análisis con expresiones proposicionales que reflejan condiciones entre los elementos, y diagramas de Venn para visualizar las operaciones. Esto permitió reforzar el vínculo entre la lógica matemática y su implementación computacional.

Desde lo organizativo, el proyecto promovió la colaboración entre ambos integrantes, la división clara de tareas, la validación cruzada del código y una documentación consistente. Además, sentó las bases para comprender con mayor profundidad temas como álgebra relacional, optimización de consultas y estructuras de datos, que se abordarán en materias futuras.

En definitiva, este tipo de ejercicios integradores nos permiten articular múltiples saberes, ejercitar el pensamiento lógico y estructurado, fortaleciendo habilidades tanto técnicas como de trabajo en equipo.

Referencias bibliográficas

- *Tecnicatura Universitaria en Programación a Distancia. Programación I. Unidad 10: Búsqueda y Ordenamiento* (2025). (1° ed.). Universidad Tecnológica Nacional.
- *Tecnicatura Universitaria en Programación a Distancia. Matemática. Unidad 4: Conjuntos* (2025). (1° ed.). Universidad Tecnológica Nacional.
- Python Software Foundation. (s.f.). *Built-in types — frozenset.union*. Python 3.12. https://docs.python.org/3/library/stdtypes.html#frozenset.union

Anexo

- URL con los diagramas de Venn, conjuntos y operaciones solicitadas: https://whimsical.com/mate-integrador-dos-varelasosa-diagramas-CxuksBMLmHF23iNkY1Q9Hj
- Código completo en GitHub: https://github.com/santiagovOK/matematica_integracion-dos_VarelaSosa/blob/main/matematica_integracion-dos_VarelaSosa.py
- Repositorio en GitHub: https://github.com/santiagovOK/matematica_integracion-dos_VarelaSosa/tree/main
- URL del video explicativo en YouTube: https://youtu.be/uUo-6G-x5v0

Codigo completo:

```
from datetime import date

def ingresar_dnis():

    pregunta = input("Querés ingresar tus DNIs (I) o usamos los de ejemplo (E)? Responde con I o E: ").upper().strip()

    if pregunta == "E":
        lista_dnis = ["26931802","28389594"]
    elif pregunta == "I":
        lista_dnis = input("Ingresá los DNIs separados por coma, sin espacios y sin puntos: ").split(",")
    else:
        print("Opción no válida. Usando DNIs de ejemplo.")
        lista_dnis = ["26931802","28389594"]

    # Imprimimos la lista de DNIs ingresados
    print("\nDNIs ingresados: ", lista_dnis, "\n")

    return lista_dnis

def contar_frecuencia_digitos(dni):
    # Creamos un diccionario para almacenar frecuencias de cada dígito
    frecuencias = {}
    for digito in dni:
        if digito in frecuencias:
            frecuencias[digito] += 1
        else:
            frecuencias[digito] = 1
```

```
    return frecuencias

# Convertimos el DNI a conjunto de dígitos únicos
def obtener_digitos_unicos(dni):
    return set(dni)

"""

Agregar la función bubble_sort() que usamos para el programa de programación I, así ordenamos los resultados de unión, intersección, diferencia simétrica, etc.)

"""

def bubble_sort(lista):
    # Obtenemos la longitud de la lista.
    longitud = len(lista)
    # Establecemos un bucle for para iterar sobre la lista.
    # El rango será la longitud de la lista, por más que el ordenamiento se complete antes. Sino deberíamos usar `break` y es mala práctica.
    for pasada in range(longitud):
        # Realizamos las comparaciones y los intercambios necesarios en la lista para ordenarla.
        for i in range(0, longitud - pasada - 1):
            # Comparamos los elementos adyacentes y los intercambiamos si están en el orden incorrecto.
            if lista[i] > lista[i + 1]:
                lista[i], lista[i + 1] = lista[i + 1], lista[i]
    # Retornamos la lista ordenada.
    return lista

def analizar_dnis(lista_dnis):
    # Establecemos listas vacías para almacenar los datos posteriores
    conjuntos_digitos = []
    frecuencias_digitos = []
    sumas_digitos = []

    # Procesamos cada DNI para las operaciones
    for dni in lista_dnis:
        conjunto = obtener_digitos_unicos(dni)
        conjuntos_digitos.append(conjunto)
        frecuencias_digitos.append(contar_frecuencia_digitos(dni))
        suma = sum(int(digito) for digito in dni)
        sumas_digitos.append(suma)
```

```
    # Realizamos las operaciones de conjuntos a la vez que ordenamos los resultados con
bubble_sort
    union = bubble_sort(list(set.union(*conjuntos_digitos)))
    interseccion = bubble_sort(list(set.intersection(*conjuntos_digitos)))

    # Inicializamos la diferencia simétrica como un conjunto vacío
    diferencia_simetrica = set()

    # Calculamos la diferencia simétrica entre todos los pares
    for conjunto1 in range(len(conjuntos_digitos)):
        for conjunto2 in range(conjunto1 + 1, len(conjuntos_digitos)):
            diferencia_simetrica_actual = conjuntos_digitos[conjunto1] ^
conjuntos_digitos[conjunto2]
            for elemento in diferencia_simetrica_actual:
                diferencia_simetrica.add(elemento)

    # Ordenamos la diferencia simétrica con bubble_sort
    diferencia_simetrica = bubble_sort(list(diferencia_simetrica))

    # Mostramos los resultados
    print(f"Unión: {union}")
    print(f"Intersección: {interseccion}")
    print(f"Diferencia Simétrica: {diferencia_simetrica}")
    print(f"Suma de dígitos por DNI: {sumas_digitos}")
    print(f"Frecuencia de dígitos: {frecuencias_digitos}")


    # Evaluamos la Relación equilibrada entre pares de conjuntos de dígitos en base a la
expresión lógica elegida.
    print("\nEvaluación de la condición lógica (expresión lógica sobre relaciones equilibradas)
entre los conjuntos de dígitos: ")

    # Calculamos la intersección: dígitos compartidos por ambos DNIs
    for indice_a in range(len(conjuntos_digitos)):
        # Iteramos sobre los conjuntos restantes para comparar con el conjunto actual
        for indice_b in range(indice_a + 1, len(conjuntos_digitos)):
            conjunto_a = conjuntos_digitos[indice_a]
            conjunto_b = conjuntos_digitos[indice_b]

            # Calculamos la intersección: dígitos compartidos por ambos DNIs (intersección A ∩ B)
            interseccion = conjunto_a & conjunto_b

            # Calculamos la unión: todos los dígitos distintos entre ambos DNIs (unión A ∪ B)
            union = conjunto_a | conjunto_b
```

```
            # Establecemos una condición para una relación equilibrada: más de 3 compartidos Y al
menos 8 distintos
            if len(interseccion) > 3 and len(union) >= 8:
                print(f"\nRelación equilibrada entre DNI {indice_a + 1} y DNI {indice_b + 1}\n")
            else:
                print(f"\nNo hay relación equilibrada entre DNI {indice_a + 1} y DNI {indice_b + 1}\n")

def es_bisiesto(anio):
    if anio % 4 == 0:
        if anio % 100 != 0 or anio % 400 == 0:
            return True
    return False

def analizar_anios():
    entrada = input("Ingrese los años de nacimiento separados por coma (sin espacios):
").split(",")
    lista_anios = []

    # Convertimos las cadenas (cada año ingresado) a enteros
    for a in entrada:
        lista_anios.append(int(a))

    print("Años ingresados:", lista_anios)

    # Almancenamos en variables los resultados de las funciones
    todos_despues_2000 = True
    alguno_bisiesto = False

    # Recorremos la lista de años para verificar las condiciones
    for anio in lista_anios:
        if anio <= 2000:
            todos_despues_2000 = False
        if es_bisiesto(anio):
            alguno_bisiesto = True

    # Imprimimos los resultados según las condiciones
    # Si todos los años son mayores a 2000, imprimimos "Grupo Z"
    if todos_despues_2000:
        print("Grupo Z")

    # Si alguno de los años es bisiesto, imprimimos "Tenemos un año especial"
    if alguno_bisiesto:
        print("Tenemos un año especial")
```

```
    # Calculamos las edades de las personas nacidas en los años ingresados

    # Obtenemos el año actual con date.today().year
    anio_actual = date.today().year
    # Calculamos las edades restando el año actual a cada año de nacimiento
    edades = [anio_actual - anio for anio in lista_anios]

    print("Edades calculadas:", edades)

    # Calculamos el producto cartesiano entre los años y las edades
    print("Producto cartesiano (año × edad):")
    for anio in lista_anios:
        for edad in edades:
            print(f"({anio}, {edad})")

def main():
    # Bandera para controlar el bucle principal
    bandera = True

    # Bucle principal del programa
    while bandera:
        # Mostramos el mensaje de bienvenida
        print("\nBienvenido al programa\n")
        # Llamamos a la función para ingresar los DNI
        lista_dnis = ingresar_dnis()
        # Llamamos a la función para analizar los DNI
        analizar_dnis(lista_dnis)
        # Llamamos a la función para analizar los años de nacimiento
        analizar_anios()


        # Solicitar opción al usuario para continuar o terminar
        opcion = input("Si querés continuar, ingresa 'S', si no, ingresa 'N' y el programa terminará:
").upper().strip()

        if opcion == "N":
            bandera = False
            print("\nEl programa finalizó.\n")
        elif opcion != "S":
            print("Opción no válida. Por favor, ingresa 'S' o 'N'.")


if __name__ == "__main__":
```

main()