COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI
DEPARTAMENTO DE MATEMÁTICA APLICADA

TITULACIÓN

Grado en Ingeniería Matemática e Inteligencia Artificial

MATEMÁTICA DISCRETA

Curso 2022-2023

PRÁCTICA 1: ARITMÉTICA MODULAR

OBJETIVOS GENERALES

- Construir una librería "modular.py" que proporcione un conjunto amplio de utilidades para el tratamiento de expresiones y la resolución de ecuaciones en aritmética modular.
- Medir y analizar el rendimiento de los algoritmos y optimizarlos para buscar alternativas más eficientes para hacer que los programas pedidos se ejecuten en el menor tiempo posible.
- Integrar los módulos anteriores en un sistema interactivo "IMAT-LAB" de resolución de ecuaciones en aritmética modular.

DESCRIPCIÓN FUNCIONAL DETALLADA DEL PROYECTO

El objetivo principal de este primer proyecto es desarrollar en Python una librería de utilidades de aritmética modular y un programa interactivo de utilidades aritméticas **"IMAT-LAB"** que proporcione una interfaz que permita al usuario interactuar de forma cómoda con la librería y resolver diversos tipos de problemas aritméticos.

El programa de resolución de ecuaciones "IMAT-LAB" funcionará mediante el uso de una lista de "comandos" predefinidos. Cada "comando" recibirá varios parámetros que definirán un cierto problema aritmético (calcular máximo común divisor, resolver inversa módulo p, resolver un sistema de congruencias...). El sistema tendrá dos modos de funcionamiento:

- **Modo interactivo:** Una interfaz de usuario que permita escribir comandos y que, cada vez que el usuario introduzca un comando, calcule y muestre en pantalla la solución de los correspondientes problemas en tiempo real.
- **Modo de procesamiento por lotes ("batch"):** Un sistema de procesamiento de ficheros en el que el programa recibe como entrada un fichero que contiene una lista completa de "comandos" y devuelve en otro fichero especificado por el usuario el resultado de resolver cada uno de los problemas indicados (como si un usuario hubiera introducido uno por uno los problemas del fichero en el modo interactivo).

En ambos casos, el programa se ejecutará hasta que lea una lína vacía ("\n") y, en tal caso, finalizará la ejecución.

Se implementarán como mínimo dos scripts, sin perjuicio de que cada grupo de prácticas quiera añadir cualquier módulo adicional que necesite para implementar sus algoritmos:

- **"modular.py"**: Librería principal de cálculo aritmético, que realiza el trabajo de resolución de los problemas aritméticos.
- **"imatlab.py"**: Interfaz de acceso interactivo o por lotes a la librería anterior. Si este script se ejecuta sin parámetros, lanzará la interfaz de usuario para el modo interactivo. Este script deberá además contener una función:

$$run\_commands(fin : TextIO, fout : TextIO)$$

que reciba dos manejadores de ficheros de texto fin y $fout$, de entrada y salida ya abiertos para lectura y escritura respectivamente y ejecute línea por línea los comandos proporcionados por fin, escribiendo los resultados en $fout$. Si fin y $fout$ no corresponden con la entrada y salida estándar, esta función deberá ejecutar el modo de procesamiento por lotes para la entrada fin, guardando el resultado en el fichero $fout$.

LISTA DE COMANDOS

De forma general, todos los comandos de IMAT-LAB devolverán el código de error "NE" (No Existe) si el problema requerido no tiene solución y devolverán "NOP" si hay algún error en la llamada al comando, el comando no existe o no ha sido implementado. En ningun caso el programa general ("imatlab.py") debe sufrir un error por una invocacion incorrecta de un comando o por la no existencia de solución a uno de los problemas planteados.

Se proporciona un fichero "ejemplosComandos.txt" con ejemplos de utilización de los comandos y un fichero "ejemplosSalida.txt", resultado de ejecutar "imatlab.py" por lotes sobre "ejemplosComandos.txt", que muestra lo que debería dar cada uno de ellos como resultado.

Como tarea opcional, en el modo interactivo, en lugar de estos dos códigos, la interfaz puede mostrar un mensaje de error dando detalles del tipo de error que se ha producido ("p no es un entero", "no existe ninguna solución", "p debe ser un número primo", etc.). Para ello, la librería modular.py deberá implementar un correcto sistema de gestión de excepciones (implementando alguna excepción personalizada si fuera necesario).

Nota importante: No se permitirá cambiar los prototipos de las funciones aquí detalladas. Los grupos de prácticas tienen libertad para modificar y ampliar la librería "modular.py" con cuantas funciones auxiliares sean necesarias para completar las funcionalidades pedidas (e incluso crear otros módulos, si lo creen conveniente), pero, en última instancia, la interfaz de funciones de la librería deberá ser la establecida en este documento.

1. Implementar en "modular.py" una función

$$es\_primo(n : int) \longrightarrow bool$$

que reciba un entero n y devuelva verdadero si es un número primo y falso en caso contrario. Utilizándola, agregar a IMAT-LAB un comando

$$primo(n)$$

que escriba "Sí" si n es primo y "No" en caso contrario.

2. Implementar en "modular.py" una función

$$lista\_primos(a : int, b : int) \longrightarrow List[int]$$

que reciba dos enteros a y b y devuelva la lista de números primos en el intervalo $[a, b)$. Utilizándola, agregar a IMAT-LAB un comando

$$primos(a, b)$$

que realice esta funcionalidad y devuelva la lista de primos obtenida separados por comas:

$$p_1,\, p_2,\, \ldots,\, p_n$$

Si no existe ningún primo en el intervalo solicitado, el comando "primos" devolverá "NE" (o un mensaje de error adecuado en el modo interactivo).

3. Implementar en "modular.py" una función

$$factorizar(n : int) \longrightarrow Dict[int, int]$$

que reciba un entero n y devuelva un diccionario cuyas claves sean los primos que dividan a n y sus valores los correspondientes exponentes en la descomposición en producto de factores primos de n. Utilizándola, agregar a IMAT-LAB un comando

$$factorizar(n)$$

que realice dicha funcionalidad e imprima el correspondiente diccionario en la forma:

$$p_1 \colon e_1,\, p_2 \colon e_2,\, \ldots,\, p_k \colon e_k$$

donde $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.

4. Implementar en "modular.py" una función

$$mcd(a : int, b : int) \longrightarrow int$$

que reciba dos enteros a y b y devuelva su máximo común divisor. Así mismo, programar una función

$$bezout(a : int, b : int) \longrightarrow Tuple[int, int, int]$$

que reciba dos enteros a y b y devuelva (d, x_0, y_0), donde $d = (a, b)$ y (x_0, y_0) satisface

$$d = ax_0 + by_0$$

Utilizando estas funciones, agregar a IMAT-LAB un comando

$$mcd(a, b)$$

que calcule el máximo común divisor.

5. (Opcional) Agregar a "modular.py" una función

$$mcd\_n(nlist : List[int]) \longrightarrow int$$

que calcule el máximo común divisor de una lista de enteros $(a_1, \ldots, a_n)$ y una función

$$bezout\_n(nlist : List[int]) \longrightarrow Tuple[int, List[int]]$$

que devuelva $(d, (x_1, \ldots, x_n))$, donde $d = (a_1, \ldots, a_n)$ y $(x_1, \ldots, x_n)$ es una solución de la ecuación

$$d = a_1x_1 + \ldots + a_nx_n$$

Utilizar estas funciones para expandir la funcionalidad del comando mcd en IMAT-LAB para que admita cualquier número de enteros

$$mcd(a_1, \ldots, a_n)$$

y devuelva su máximo común divisor.

6. Implementar en "modular.py" una función

$$coprimos(a : int, b : int) \longrightarrow bool$$

que reciba dos enteros a y b y devuelva si son coprimos. Utilizándola, agregar a IMAT-LAB un comando

$$coprimos(a, b)$$

que realice esta funcionalidad y devuelva "Sí" si son coprimos y "No" en caso contrario.

7. Implementar en "modular.py" una función

$$potencia\_mod\_p(base : int, exp : int, p : int) \longrightarrow int$$

que reciba tres enteros $base$, exp y p y devuelva $base^{exp} \pmod p$. Usando dicha función, agregar a IMAT-LAB un comando

$$pow(base, exp, p)$$

que realice esta funcionalidad.

8. Implementar en "modular.py" una función

$$inversa\_mod\_p(n : int, p : int) \longrightarrow int$$

que reciba dos enteros n y p y devuelva el inverso de n módulo p. Usando dicha función, agregar a IMAT-LAB un comando

$$inv(n, p)$$

que realice esta funcionalidad.

9. Implementar en "modular.py" una función

$$euler(n : int) \longrightarrow int$$

que reciba un entero n y calcule la función de Euler $\varphi(n)$. Usando dicha función, agregar a IMAT-LAB un comando

$$euler(n)$$

que realice esta funcionalidad.

10. Dado un entero n y un número primo p, el símbolo de Legendre, denotado $\left(\frac{n}{p}\right)$, vale

$$\left(\frac{n}{p}\right) = \begin{cases} 0 & n \equiv 0 \pmod{p} \\ 1 & n \text{ es un cuadrado módulo } p \\ -1 & \text{en caso contrario} \end{cases}$$

Investigar las propiedades del símbolo de Legendre e implementar en "modular.py" una función

$$legendre(n : int, p : int) \longrightarrow int$$

que lo calcule. Usando dicha función, agregar a IMAT-LAB un comando

$$legendre(n, p)$$

que realice esta funcionalidad.

11. Implementar en "modular.py" una función

$$resolver\_sistema\_congruencias(alist : List[int], blist : List[int], plist : List[int]) \longrightarrow Tuple[int, int]$$

que reciba tres listas de números enteros $(a_1, \ldots, a_n)$, $(b_1, \ldots, b_n)$ y $(p_1, \ldots, p_n)$, con p_k coprimos dos a dos y devuelva (r, m), donde $x \equiv r \pmod{m}$ es la solución del sistema de congruencias:

$$\begin{cases} a_1 x \equiv b_1 \pmod{p_1} \\ a_2 x \equiv b_2 \pmod{p_2} \\ \quad \vdots \\ a_n x \equiv b_n \pmod{p_n} \end{cases}$$

Usando dicha función, agregar a IMAT-LAB un comando

$$resolverSistema([a_1; b_1; p_1], [a_2; b_2; p_2], \ldots, [a_n; b_n; p_n])$$

que reciba una lista de entradas de la forma $[a_k; b_k; p_k]$ (nótese la separación con comas entre entradas/ecuaciones y la separación usando puntos y comas entre los números a_k, b_k y p_k para cada k), donde se asume que los números p_k son coprimos dos a dos, y escriba la solución del correspondiente sistema de ecuaciones en la forma: $r(modm)$

12. (Opcional) Expandir la funcionalidad de *resolver_sistema_congruencias* y del comando *resolverSistema* para resolver sistemas de ecuaciones donde los módulos no son coprimos entre sí.

13. Investigar el algoritmo de Cipolla y programar en "modular.py" una función

$$raiz\_mod\_p(n : int, p : int) \longrightarrow int$$

que reciba un entero n y un número primo p y calcule una raíz de n módulo p, es decir, un entero x tal que

$$x^2 \equiv n \pmod{p}$$

Usando dicha función, agregar a IMAT-LAB un comando

$$raiz(n, p)$$

que devuelva lo siguiente:

- Si n tiene dos raíces distintas $x_1 < x_2$, las escribe en orden: "x_1, x_2"
- Si n tiene una única raíz x, escribe "x"
- Si n no tiene raíces, como con el resto de comandos, escribe "NE" en modo "batch" o un mensaje de error adecuado en modo interactivo.

14. Usando la función anterior, implementar en "modular.py" una función

$$ecuacion\_cuadratica(a : int, b : int, c : int, p : int) \longrightarrow Tuple[int, int]$$

que reciba enteros a, b, c y p, con p número primo, y devuelva las dos soluciones módulo p (posiblemente repetidas) de la ecuación

$$ax^2 + bx + c \equiv 0 \pmod{p}$$

Usando dicha función, agregar a IMAT-LAB un comando

$$ecCuadratica(a, b, c, p)$$

que implemente la funcionalidad anterior y devuelva lo siguiente

- Si la ecuación tiene dos raíces distintas $x_1 < x_2$, las escribe en orden: "x_1, x_2"
- Si la ecuación tiene una única raíz doble x, escribe "x"
- Si la ecuación no tiene raíces, como con el resto de comandos, escribe "NE" en modo "batch" o un mensaje de error adecuado en modo interactivo.

15. (Opcional) Complementar los comandos de IMAT-LAB añadiendo uno o más comandos nuevos que resuelvan algún problema matemático de aritmética o aritmética modular que no se encuentre entre los anteriores, implementando, para ello, las correspondientes nuevas funcionalidades en la librería "modular.py" (u otras librerías adicionales, si fuera necesario). Para ejemplificar los nuevos comandos agregados, escribir un fichero de entrada "nuevosComandos.txt", en el que se muestren ejemplos de uso de los nuevos comandos implementados.

BENCHMARKING Y PROFILING

Se deberá intentar que las funciones anteriormente descritas se ejecuten de la forma más eficientemente posible. Para medir dicha eficiencia, se ejecutarán diversos bancos de pruebas ("benchmarks") cuyos tiempos de ejecución serán medidos por la librería "timeit".

En Moodle pueden encontrarse enlaces a tutoriales sobre la librería "timeit" y la librería "cprofile", incluyendo una introducción general al uso de profilers. El "profiling" o perfilación del código sirve para analizar el tiempo de ejecución que una función consume en cada una de sus subrutinas. De esta manera se puede detectar si existe algún "cuello de botella" en alguna función interna que pueda estar degradando el rendimiento global y, al saber qué funciones tienen un mayor número de llamadas o un mayor tiempo total de ejecución, detectar aquellos módulos que merece más la pena optimizar si se desea mejorar el tiempo total de ejecucion.

Se proporcionan varios ficheros ("benchmarks") con muestras seleccionadas de comandos de IMAT-LAB, junto con un conjunto de tiempos de ejecución de referencia para cada fichero.

Se proporciona, así mismo, un script "imatlab_benchmark.py" que permite hacer medición de tiempos y profiling sobre el modo por lotes de "imatlab.py". Con el fin de hacer la medición de tiempos equiparable para todos los alumnos, la toma de tiempos final se realizará con este script y se ejecutará en los ordenadores de los laboratorios de ICAI. Con el fin de permitir a los estudiantes trabajar en la práctica sin necesidad de estar presencialmente en ICAI, los alumnos pueden acceder en remoto a estos ordenadores en cualquier momento a través de un navegador mediante el sistema de aulas virtuales de ICAI. En Moodle se puede encontrar un manual para el acceso remoto y uso de las aulas virtuales.

Para intercambiar código y datos entre estos ordenadores y un ordenador local, lo más práctico es utilizar cualquier forma de almacenamiento en nube (por ejemplo, el OneDrive asociado a las cuentas de Comillas) y acceder a él tanto desde vuestro ordenador como desde el aula virtual.

TIEMPOS DE REFERENCIA

Lista de tiempos de referencia para cada "benchmark":

Fichero	**Bronce**	**Plata**	**Oro**
primosTest.txt	10 s	4.3s	730 ms
factorTest.txt	7.5 s	2.33 s	1.99 s
mcdTest.txt	2.7 s	1.5 s	1 s
potenciaTest.txt	800 ms	220 ms	120 ms
invTest.txt	6.5 s	4.2 s	3.125 s
eulerTest.txt	5.5 s	1,06 s	779 ms
sistemaTest.txt	4.5 s	1.5 s	988 ms
cuadraticaTest.txt	1.1 s	355ms	218 ms

ENTREGAS

Como resultado final de la práctica se entregará

- El código desarrollado, apropiadamente comentado y documentado (ver guía de estilo PEP8 en Moodle):
  - La librería "modular.py".
  - El programa "imatlab.py".
  - Si el grupo ha implementado algún comando adicional, un fichero de ejemplo "nuevosComandos.txt" que incluya algunos ejemplos significativos de ejecución de los nuevos comandos implementados.
  - Cualquier otra librería, script o fichero de configuración adicional que haya sido programada por los alumnos para el desarrollo de la práctica y que sea necesario para ejecutar los scripts.
- Una memoria en PDF con el nombre P1GPxx.pdf (donde GPxx es el número de grupo de la pareja) en el que se especificarán, como mínimo, los siguientes elementos:
  - Nombre y apellidos de los integrantes del grupo y código del grupo de prácticas (GPxx).
  - Decisiones y estrategias generales de diseño tomadas (si se han creado librerías adicionales, cómo se ha agrupado la funcionalidad, etc.)
  - En cuanto al diseño general del sistema de resolución "IMAT-LAB":
    - Esquema de diseño de la aplicación (descripción de los nuevos módulos implementados y su organización).
    - Si se hubiera implementado, explicación de la forma en que se han gestionado los mensajes de error individualizados para los comandos.
    - Descripción de funcionalidades adicionales, si las hubiera, implementadas por el grupo, explicando aquellos algoritmos adicionales que se hayan implementado.
  - Para cada tipo de problema resuelto por la librería "modular.py":
    - Descripción del algoritmo utilizado.
    - Justificación matemática (incluyendo las demostraciones o citas a teoremas cuando sea necesario), de que el algoritmo utilizado resuelve el problema pedido.
    - Explicación de las estrategias utilizadas para optimizar el rendimiento del código en dicho problema.
    - Cualquier comentario que se considere relevante sobre el diseño de la implementación.
  - Bibliografía, incluyendo referencias debidamente citadas a cualquier fuente utilizada para desarrollar la práctica.

Todos los ficheros anteriores se cargarán en la tarea correspondiente de Moodle.

CRITERIOS DE EVALUACIÓN

- **Diseño general del programa IMAT-LAB (1 punto):** Se puntuará el diseño general del programa, la implementación correcta de la entrada/salida y la gestión correcta de los casos y mensajes de error. El programa deberá ejecutarse sin bugs, independientemente de la entrada introducida por el usuario (ya sea a través de un fichero o en el modo interactivo). Se valorará positivamente la implementación de mensajes de error individualizados para cada función en el modo interactivo como elemento de calidad de la interfaz del sistema.
- **Implementación de las funciones solicitadas en "modular.py" e implementación de los comandos correspondientes en el programa IMAT-LAB (6 puntos):** Se sumará la puntuación de cada una de las funciones implementadas (0.5 puntos por comando obligatorio). Se requerirá que las funciones se ejecuten sin errores en todos los casos solicitados y que se realice un control correcto de excepciones. Se puntuará positivamente la implementación de las funciones y comandos adicionales opcionales (hasta +1 punto).
- **Eficiencia del código (1 punto):** Para cada uno de los ficheros de prueba se ejecutará el programa IMAT-LAB del grupo de prácticas usando el script "imatlab_benchmark.py" proporcionado en Moodle. Según el intervalo en que se encuentre el tiempo medido, se obtendrá una puntuación de la siguiente manera:
  - Si el tiempo de ejecución del fichero está por encima del tiempo de bronce (máximo): 0 puntos.
  - Alcanzar la marca de bronce: +0.1 puntos.
  - Alcanzar la marca de plata: +0.2 puntos.
  - Alcanzar la marca de oro: +0.3 puntos
  - Si un algoritmo consigue una mejora excepcional respecto a la marca de oro, se podrá puntuar con +0.5 puntos.
- **Organización y calidad del código (0.5 puntos):** Se valorará el uso de una programación estructurada, siguiendo un buen estilo de programación, la modularidad y reutilización del código, así como que el código esté correctamente documentado.
- **Memoria (1.5 puntos):**
  - Documentación de las decisiones de diseño y la estructura de módulos: 0.5 puntos
  - Justificaciones teóricas de los algoritmos utilizados y descripción de las estrategias de optimización: 1 punto