INFORME DEL PROYECTO DE PROGRAMACIÓN LÓGICA

Inteligencia Artificial

MARCOS PADILLA HERRERA

❖ Idea principal del proyecto

El proyecto realizado es un clasificador de números. El objetivo principal del mismo es que se indiquen ciertos detalles acerca de un número que el usuario introduce por teclado. Estos detalles serán subdivididos en dos bloques; características y naturaleza.

❖ Funcionalidades

Una vez el usuario haya introducido un número, el programa proporcionará la siguiente información:

- Visualización del número introducido
- Si el número es par o impar
- Si el número es primo
- Si el número es omirp
- Si el número es perfecto
- Si el número es feliz
- Si es un número natural
- Si es un número entero
- Si es un número racional
- Si es un número irracional
- Si es un número real
- Si es un número complejo

Además, al final de estas características se dispone de una guía de ayuda para indicar qué significa cada campo.

❖ Contenido del código

En primer lugar, hay que indicar que el programa está dividido en dos partes; la primera indica las características del número introducido, mientras que la segunda indica la naturaleza del mismo. Aunque en la práctica hemos introducido todos los hechos juntos, y hemos realizado el programa de manera conjunta, explicaremos los dos subprogramas por separado.

❖ Restricciones

Como comentaremos más adelante, para realizar ciertas comprobaciones es necesario que el número que va a ser comprobado sea guardado en una variable de tipo string, para que más adelante podamos guardar cada elemento del string en una lista y así poderlos manipular por separado mediante la función split_string. Por ello, para el correcto funcionamiento del programa es necesario que el usuario escriba el número entre comillas dobles y con un espacio entre cada una de las cifras; por ejemplo, el número 523 se escribiría así: "5 2 3". Ya con el número recogido de esta manera, el programa realizará las comprobaciones necesarias. Cuando el programa ya no necesite el número almacenado en una lista, lo transformará a un string nuevamente con la función list_to_string, de modo que ya aparecerán todos los números juntos, y además, saldrá visualizado por pantalla correctamente. Por último se ha implementado la función long, que devuelve el número de dígitos que contiene el string con el número introducido; si ese valor sobrepasa los 3 dígitos el programa no dejará continuar, ya que solo se puede introducir hasta el 999.

❖1ª parte: características.

En esta primera parte, nuestro programa pedirá al usuario un número entero por teclado, entre el 0 y el 999; es decir, el número será obligatoriamente entero y positivo, y no será mayor que 999. A continuación, se indicará por pantalla si el número introducido es par, impar, primo, omirp, perfecto o feliz. La razón por la que el número debe ser entero y positivo es simple: un número decimal o fraccionario no puede ser clasificado como par o impar, ni puede ser primo, omirp, feliz o perfecto. Por tanto, sería inútil poder introducir números decimales.

En cuanto a los negativos, si introdujéramos un número inferior a 0 solo nos serviría para indicar si este es par o impar; para el resto de funcionalidades, ocurriría lo mismo que con los números decimales. Por último, hemos limitado el número máximo a 999, ya que si lo hiciéramos con un rango más amplio, la base de datos de números primos, perfectos y felices sería muy extensa. No obstante, si introdujésemos un número mayor que 999, nos indicaría correctamente si es par o impar, ya que para realizar esta comprobación no hacemos uso de ninguna base de datos.

Finalmente, cuando el programa ya ha visualizado por pantalla todos estos aspectos, aparecerá una guía de ayuda con las definiciones que comentaremos a continuación.

Ayuda.

Aunque en la ejecución del programa la ayuda aparece en último lugar, explicaremos este punto desde el principio para facilitar la comprensión del resto del programa. En este apartado definimos las características que vamos a tener en cuenta para la clasificación de los números. Así, aclaramos los siguientes conceptos:

- Par: todo número que, al dividirlo entre 2, el resto da como resultado 0.

- Impar: todo número que, al dividirlo entre 2, el resto da como resultado 1.
- Primo: todo número que solo puede dividirse entre sí mismo y entre 1, dando un resto igual a 0.
- Omirp: todo número primo que, al invertir sus cifras, da como resultado otro número primo.

- Perfecto: todo número que es igual a la suma de sus divisores propios positivos.
- Feliz: todo número que cumple que, cuando se suman los cuadrados de sus dígitos y se repite el proceso cuantas veces sea necesario, se obtiene un 1 en algún momento.

Este puede ser un apartado especialmente útil para aquellos usuarios que no estén familiarizados con esta terminología, sobre todo con los números felices y los números perfectos.

Hechos.

Como punto de partida, hemos definido una serie de hechos, sobre los que trabajaremos a lo largo del programa. Así, hemos introducido un hecho por cada número primo existente entre el 0 y el 999. De esta forma, lo único que deberá hacer el programa para ver si el número X es primo será comprobar si hay definido algún hecho con ese número. Es por ello que decidimos realizar esta base de datos solo hasta el número 999, ya que la infinidad de números primos dificulta la realización de esta base de datos. Por lo tanto, si el usuario introduce un número y el programa indica que no es primo, significa que no hay ningún hecho con ese número, o que el usuario ha introducido un número fuera del rango establecido (menor que 0, mayor que 999 ó decimal).

Asimismo, hemos realizado la misma tarea para los números felices y los números perfectos; en este último caso es especialmente útil escribir un hecho por cada número, ya que la escasa cantidad de estos números implica que sea mucho más sencillo definir todos los hechos que realizar la comprobación específica (habría que ir sumando todos sus divisores propios positivos).

A partir de estos hechos el programa hará las comprobaciones necesarias con el número que sea introducido por teclado.

Funcionamiento.

Definimos una serie de funciones para cada una de las características previamente descritas. De este modo, comprobamos que el número introducido por pantalla sea primo, por ejemplo, y dependiendo de la respuesta, el programa imprimirá por pantalla un mensaje afirmativo o negativo. Procederemos del mismo modo para comprobar si el número es perfecto o feliz. Para comprobar si el omirp, previamente invertimos el número en concreto, con el uso de una lista auxiliar y la función invertir. Por último, para comprobar si el número es par o impar, primero utilizamos la función last para guardar el último dígito del número, y sobre este realizar la división entre 2 y comprobar así si es par o impar.

Evidentemente, esta es la implementación de los métodos que necesitaremos más adelante; es decir, estas funciones solo realizarán su cometido cuando sean llamadas. En la parte final del código encontramos un menú; es en este punto cuando el programa pide al usuario el número por teclado, lo almacena en una variable y llama una por una a todas las funciones descritas anteriormente, para hacer las comprobaciones oportunas. Las funciones están preparadas para actuar sobre el número en cuestión, e imprimir por pantalla un mensaje u otro dependiendo del resultado obtenido.

Hay que destacar que para algunas funcionalidades guardamos el número introducido por teclado en una variable, mientras que en otras ocasiones lo guardamos en una lista. En este último caso, el fin es poder tener un acceso más cómodo a un dígito específico del número.

❖2ª parte: naturaleza.

En esta segunda parte de la práctica, el programa pedirá al usuario un número cualquiera. En este caso, no estará limitado de ninguna manera; puede ser un número positivo o negativo, decimal o entero, mayor de 999, etc. No es necesario limitar el número, ya que solo contamos con una pequeña base de datos para los números irracionales. Sin embargo, sigue siendo necesario introducir el número de la misma manera, es decir, entre comillas dobles y con un espacio entre cada dígito del mismo.

Una vez introducido el número, nos aparecerá la información acerca de la naturaleza del mismo, indicando si es natural, entero, racional o irracional, real o complejo. Por último, al final de la visualización aparecerá una guía que indica el significado de cada conjunto, tal y como se expone a continuación.

Ayuda.

En este segundo subprograma también encontramos una sección de ayuda, pero diferenciada de la primera. Es decir, dependiendo de la opción que introduzcamos (características o naturaleza), nos saldrá una guía u otra. En este caso, esta ayuda nos muestra las definiciones referentes a la naturaleza del número, tal y como expondremos a continuación:

- Natural: conjunto al que pertenecen los números enteros positivos, desde 0 a ∞.
- Entero: conjunto al que pertenecen todos los números enteros, positivos y negativos, desde -∞ a +∞. Incluye a los números naturales.
- Racional: conjunto al que pertenecen todos los números positivos y negativos, desde -∞ a +∞, que sean fraccionarios y con un número finito de decimales.
- Irracional: conjunto al que pertenecen los números fraccionarios o decimales con infinitas cifras decimales. Por ejemplo, el número e, pi, o las raíces cuadradas de números como el 2.
- Real: conjunto al que pertenecen todos los números descritos anteriormente.
- Complejo: conjunto al que pertenecen los números que están formados por una parte real y una imaginaria. Por ejemplo: 4 + 6i.

Hechos.

Al igual que en la primera parte de la práctica, hemos desarrollado una serie de hechos para establecer el conjunto de los números irracionales. Solo hemos añadido 3 números a este conjunto, quizás los más significativos (número e, número pi y número áureo), ya que ponerlos todos hasta el 1000 implicaría una base de datos demasiado extensa.

En este segundo subprograma no necesitamos más hechos; para realizar las comprobaciones del resto de conjuntos no van a ser necesarios más hechos.

Funcionamiento.

Definimos una serie de funciones que van a realizar las comprobaciones necesarias para clasificar la naturaleza del número. Así, definimos una función para cada conjunto posible en nuestro programa, de modo que imprima por pantalla un mensaje u otro dependiendo de

si el número pertenece o no a ese conjunto. El procedimiento es igual que en la primera parte de la práctica; establecemos las funciones y posteriormente las llamaremos, cuando sea necesario, y en el orden que nos convenga.

En esta ocasión vamos a hacer uso del predicado member. Sin embargo, no podemos usarlo para ver si un número pertenece a un conjunto determinado. Usaremos este predicado para comprobar si dentro de la lista donde tenemos almacenado el número aparece además algún símbolo no alfanumérico que nos ayude a clasificar el número. Por ejemplo, si encontramos el símbolo "-" ya podemos decir que el número no es natural, o si encontramos el símbolo "." concluimos que no es un número natural ni entero. Concatenando todas estas condiciones, finalmente tendremos todos los conjuntos a los que pertenece el número y a los que no.

❖ Menú.

Cuando ejecutamos el programa nos encontramos primero con un menú. En este tendremos que indicar si queremos ver las características del número o la naturaleza del mismo, accediendo a la primera o segunda parte del programa, respectivamente. Si elegimos la primera opción, el programa solicitará un número entero entre 0 y 999, mientras que si elegimos la segunda opción, podremos introducir cualquier número. Cada una de las dos partes del programa está integrada en una única función; dentro de esta se llama a todas las funciones que hemos implementado anteriormente. Además, no solo encontramos llamadas a funciones, sino algunos mensajes que se imprimirán por pantalla para organizar mejor los datos resultantes. Por último, tanto la función de las características como la de la naturaleza llaman a su función de ayuda respectiva, que se visualizará al final de la ejecución del programa.

❖ Conclusiones.

Debido a la corta longitud del código de este proyecto, deja claro el potencial de Prolog. Por la razón de que no nos ha hecho falta programar un código muy extenso para poder realizar un clasificador de números bastante completo. Además gracias a la facilidad de lectura del lenguaje, el trabajo se hace mucho más llevadero e incluso entretenido. Si bien en este caso nos ha servido para realizar varias comprobaciones, puede llegar a ser mucho más útil para realizar cualquier otro tipo de clasificación, cálculos matemáticos, puzzles, etc.