UNIDAD III

Gestión, representación y almacenamiento de datos

Representación digital de los números

Cualquier número *N* nuestro sistema numérico, de base decimal, puede ser representado en cualquier otra base *b*, según:

$$N = \alpha_n b^n + \alpha_{n-1} b^{n-1} + \alpha_{n-2} b^{n-2} + \ldots + \alpha_1 b^1 + \alpha_0 b^0 + \beta_1 b^{-1} + \beta_2 b^{-2} + \ldots$$

donde los coeficientes α_k y β_k definen la representación del número en la base.

Para calcular estos coeficientes, separamos la parte entera (decimal) del número, dividimos (multiplicamos) por la base *b* y nos quedamos con el **resto** (operación %). Por ejemplo, la parte entera del número anterior se puede escribir de la forma:

$$I(N) = \alpha_0 + b(\alpha_1 + b(\alpha_2 + b(\alpha_3 + \ldots + b\alpha_n)\ldots)$$

de modo que: $\alpha_0 = I(N)\ \%\ b$. Nos quedamos con la diferencia:

$$I_1 = (I(N) - \alpha_0)/b = \alpha_1 + b(\alpha_2 + b(\alpha_3 + \ldots + b\alpha_n)\ldots)$$

y repetimos la operación consecutivamente hasta encontrar todos los coeficientes.

Representación digital de los números

La unidad mínima de almacenamiento de computadora (en memoria y disco) es el **bit**. Estos sólo pueden contener 1 ó 0, de modo que todos los números son trabajados en su **representación en base binaria**.

Por ejemplo, tomemos el número 132 y busquemos su representación binaria:

132	132 % 2 = 0	(132 - 0)/2 = 66
66	66 % 2 = 0	(66 - 0)/2 = 33
33	33 % 2 = 1	(33 - 1)/2 = 16
16	16 % 2 = 0	(16 - 0)/2 = 8

8	8 % 2 = 0	(8 - 0)/2 = 4
4	4 % 2 = 0	(4 - 0)/2 = 2
2	2 % 2 = 0	(2 - 0)/2 = 1
1	1 % 2 = 1	(1 - 1)/2 = 0

De modo que 132 = $(10000100)_2$, por lo tanto **requiere 8 bits** para ser representado digitalmente.

Un programa computacional necesita convertir los números cada vez que necesita **comunicarse** con los humanos.

Representación digital de los números

La representación digital de cada número está definida por el tipo de dato utilizado.

Número enteros

El tipo de dato `int` (según es definido por el lenguaje C) utiliza 32 bits (4 Bytes) para almacenar toda la información de cada variable, reservando 1 bit para el signo.
Por ejemplo, el 132 en memoria podría representarse como:

0	000 0000 0000 0000 0000 0000 1000 0100

Esto crea una limitación: este tipo de datos sólo permite representar los números en el rango:

$$[\ -2^{31},\ \ 2^{31}\ ] = [\ -2.147.483.648,\ 2.147.483.648\ ]$$

C también define más tipos de números enteros, tales como los `unsigned int` (solo enteros positivos), o los `long int` (que usan 64 bits). Esto extiende el rango de números que se pueden representar.

En **Python**, de forma **excepcional,** los números enteros cambian automáticamente la cantidad de bits utilizados según necesidad. No obstante, objetos que utilizan tipos de datos estáticos como los **arrays de numpy** tienen la mismas limitaciones que C.

La unidad usual de medición del tamaño de información almacenada en una computadora es el Byte: 1 Byte = 8 bits

Representación digital de los números

Números flotantes

Estos números se caracterizan por tener una resolución decimal que puede variar (punto flotante). Para ello, su representación se basa en la notación $M \times 2^N$, de modo que se necesita almacenar: el signo, una mantisa y un exponente (sesgado, para incorporar signo). Por ejemplo:

0	1101 0101	1010 1010 1010 1111 0000 010
Signo	N = 8 bits	M = 23 bits

De este modo, tanto la mantisa como el exponente pueden representar rangos numéricos definidos.

La mantisa M es una fracción en binario de la forma: M = +0.1 $\beta_2 \beta_3 \beta_4 \ldots \beta_{23}$, fijando $\beta_1 = 1$.
Este formato proporciona una precisión de 6 a 9 dígitos decimales significativos, y define el número máximo ($(1-2^{-24})\ 2^{128} \simeq 3.402 \times 10^{38}$) y mínimo ($2^{-126} \simeq 1.175 \times 10^{-38}$) a representar.

C introduce el tipo `double`, de mayor precisión, que utiliza 64 bits (8 Bytes) para su almacenamiento.

En **Python**, se pueden consultar detalles de sus tipos `int` y `float` con `sys.float_info` y `sys.int_info`. Este lenguaje utiliza floantes equivalentes al tipo `double` de C.

Consideraciones

- No todos los números reales dentro del rango numérico del tipo flotante pueden ser representados digitalmente de forma exacta. Esto genera errores de redondeo en operaciones aritméticas.
- La representación flotante determina un épsilon (*e*) de resolución, de tal manera que la operación $(x + 2^{-e})$ es equivalente a x.
- La representación decimal de los números almacenados digitalmente no necesariamente refleja la precisión del número.
- Los **caracteres** y sus arrays (strings) son representados de forma equivalente a números enteros de 8 bits (1 Byte), de modo que cada valor es mapeado con la tabla de caracteres ASCII.

Almacenamiento en archivos de texto.

Entendemos por archivo de texto (o ASCII) a todo archivo que consiste en un conjunto de caracteres (incluyendo espacios, tabs y saltos de línea, entre otros) ordenados de forma secuencial.

Esto considera tanto archivos específicos para contener texto (como .txt, .md, .rst, etc) como archivos utilizados por algunos programas (.csv, .xml, .html, .py, .sh, etc).

La escritura de un valor numérico en un archivo de texto, implica:

- Conversión de binario a decimal
- Posible truncación de la representación decimal (número de decimales)
- Conversión a cadena de caracteres
- Escritura en archivo de caracteres, incluyendo un delimitador.

El proceso de lectura va en orden inverso, agregando la tarea de identificación de los caracteres que representan el dato a leer.

Ejemplo:
El entero 123 456 789, que se almacena internamente en 4 Bytes, requiere 9 Bytes para ser escrito en un ASCII.