Tecnología Digital 1: Introducción a la Programación
Trabajo Práctico 1

Escuela de Negocios, UTDT

Primer semestre 2024

Antes de comenzar, se recomienda resolver el ejercicio 12 de la Guía 4, para entender el sistema de numeración binaria y cómo pasar números enteros de base 2 a base 10 y viceversa.

Dados dos números $n, m \in \mathbb{N}$ (es decir, dos enteros mayores que 0), se define la *distancia binaria* entre n y m como la cantidad de dígitos que difieren posición a posición en sus desarrollos en base 2. Por ejemplo, la distancia binaria entre 6 y 18 es 2; y la distancia binaria entre 127 y 128 es 8:

$$\texttt{bin}(6) = 00110 \qquad \texttt{bin}(127) = 01111111$$
$$\texttt{bin}(18) = 10010 \qquad \texttt{bin}(128) = 10000000$$

Notar que, en esta definición, se asume que si dos números no contienen la misma cantidad de dígitos en su representación en base 2, la más chica de ellas se completa con ceros a la izquierda.

El objetivo de este trabajo es desarrollar funciones que permitan trabajar sobre categorías que se atribuye a los números naturales en relación a la distancia binaria. A continuación, hay dos definiciones que resultan de interés para este propósito:

- **Vecino binario aledaño**. Dos números $n, m \in \mathbb{N}$ son vecinos binarios aledaños si su distancia binaria es 1.
- **Densidad binaria**. Dados $n, a, b \in \mathbb{N}$ (con $a < b$), la densidad binaria de n en $[a, b]$ es el cociente entre la cantidad de naturales del intervalo $[a, b]$ que son binarios aledaños a n y la cantidad total de naturales del intervalo $[a, b]$.

Se pide crear un programa en Python que ofrezca las siguientes funcionalidades:

- dados $n, m \in \mathbb{N}$, obtener la distancia binaria entre n y m;
- dados $n, m \in \mathbb{N}$, determinar si n y m son vecinos binarios aledaños;
- dado $n \in \mathbb{N}$, obtener la lista de los vecinos binarios aledaños a n que son menores que n, ordenados de menor a mayor;
- dados $n, a, b \in \mathbb{N}$ ($a < b$), obtener la cantidad de números en el intervalo $[a, b]$ (incluyendo los extremos) que son vecinos aledaños a n;
- dados $n, a, b \in \mathbb{N}$ ($a < b$), obtener la densidad binaria de n en $[a, b]$, con un error menor a 10^{-5}.

Para cada una de las funcionalidades requeridas, se pide:

(a) Escribir la especificación de una función que resuelva el problema.

(b) Construir un conjunto de casos de test que se considere adecuado para ilustrar y poner a prueba el comportamiento esperado.

(c) Pensar un algoritmo y llevarlo a un programa en Python que resuelva el problema.

(d) Asegurarse de que la función pasa los casos de test construidos.

(e) Para las funciones que contengan algún ciclo, demostrar que estos terminan, escribir predicados invariantes que describan el trabajo que realizan, y usarlos para mostrar que los programas son correctos. En el caso de que una función sin ciclo utilice una función auxiliar que contenga algún ciclo, solo deberán darse estas justificaciones sobre la función auxiliar.

Al ejecutar el programa se deberá desplegar un menú desde el cual seleccionar la funcionalidad a la que se desea acceder. Dependiendo de la opción elegida y los argumentos ingresados, el programa tendrá que mostrar la respuesta que corresponde. A continuación puede observarse parte de una ejecución del programa en la cual el usuario consulta si los números 6 y 18 son o no vecinos binarios aledaños:

```
Funciones disponibles
---------------------
A. Distancia binaria [n,m]
B. Son vecinos binarios aledaños [n,m]
C. Lista vecinos binarios aledaños menores [n]
D. Cantidad de aledaños en intervalo [n,a,b]
E. Densidad binaria [n,a,b]
F. Finalizar

Seleccione una opción: B
Ingrese n: 6
Ingrese m: 18
Los números 6 y 18 no son vecinos binarios aledaños.

Presione ENTER para continuar.
```

La ejecución debe concluir únicamente si se selecciona la opción `Finalizar`; caso contrario, tiene que desplegar nuevamente el menú de opciones.

En la siguiente tabla se ilustran los mensajes que debe imprimir el programa para algunas funciones y valores ingresados:[1]

Función seleccionada	Argumento(s)	Mensaje por pantalla
`Distancia binaria`	`1,1`	`La distancia binaria entre 1 y 1 es 0`
`Distancia binaria`	`6,18`	`La distancia binaria entre 6 y 18 es 2`
`Distancia binaria`	`128,127`	`La distancia binaria entre 128 y 127 es 8`
`Son vecinos binarios aledaños`	`8,12`	`Los números 8 y 12 son vecinos binarios aledaños`
`Son vecinos binarios aledaños`	`11,12`	`Los números 11 y 12 no son vecinos binarios aledaños`
`Lista de vecinos aledaños menores`	`11`	`Los números 3, 9 y 10 son los vecinos aledaños a 11 menores que 11`
`Lista de vecinos aledaños menores`	`16`	`El número 16 no tiene vecinos aledaños menores`
`Cantidad de aledaños en intervalo`	`10,8,13`	`2 números en [8,13] son aledaños a 10`
`Cantidad de aledaños en intervalo`	`32,1,31`	`0 números en [1,31] son aledaños a 32`
`Densidad binaria`	`10,8,13`	`La densidad binaria de 10 en [8,13] es aproximadamente 0.33333`
`Densidad binaria`	`32,1,31`	`La densidad binaria de 32 en [1,31] es aproximadamente 0.0`

[1] Se espera que la densidad se muestre con a lo sumo 5 decimales.

Se deben entregar los siguientes cuatro archivos:

- `distancia_binaria.py`, con el código de las funciones: `distancia_binaria`, `son_aledaños`, `aledaños_menores`, `cant_aledaños`, `densidad_intervalo` y cualquier otra función auxiliar que se defina. Es obligatorio especificar el tipo de todas las variables y funciones mediante sugerencias de tipos (*type hints*). Las especificaciones de las funciones se deben incluir con el formato de *docstrings* presentado en clase. Este archivo **solamente** debe incluir la implementación de las funciones; es decir, **no** debe ser directamente ejecutable.
- `distancia_binaria_tests.py`, con los tests de unidad de todas las funciones definidas.
- `tp1.py`, con el código principal para ejecutar el programa. Este código es el encargado de imprimir el menú, invocar a las funciones definidas en `distancia_binaria.py` y mostrar los resultados por pantalla.
- `informe.pdf`, un documento con *(i)* la justificación de los casos de test elegidos (es decir, por qué son buenos candidatos para revelar la presencia de errores); y *(ii)* las respuestas al punto (e) enunciado arriba. Incluir cualquier aclaración adicional que se considere necesaria sobre cualquier parte del trabajo. Se espera que este documento sea conciso, de no más de cuatro páginas, y en formato PDF.

La carpeta adjunta `templates` contiene archivos de ejemplo para usar como referencia.

Observaciones:

- El trabajo se debe realizar en grupos de **tres personas**. La entrega consistirá en trabajo original realizado íntegra y exclusivamente por las personas que integran el grupo.
- La fecha límite de entrega es el **domigo 21 de abril a las 23:55**. Los TPs entregados fuera de término serán aceptados hasta 48 h pasada esta fecha, pero la demora incidirá negativamente en la nota.
- Los archivos deben subirse al formulario *TP1: Entrega* de la página de la materia en el campus virtual.
- El programa entregado debe correr correctamente en `Python3`.
- Las únicas bibliotecas que se permite importar son `typing` (para *type hints*), y `unittest` (para correr los tests de `distancia_binaria_tests.py`).
- Sólo pueden usarse las instrucciones y estructuras de control vistas en clase. No pueden usarse iteradores (`for`); tampoco puede hacerse *slicing* sobre *strings*.
- Para obtener la representación binaria de un número natural, se permite usar las funciones `bin` y `replace` de Python. La expresión `bin(n)` devuelve un string con formato `'0bx'`, donde `x` es la representación en base 2 del entero `n`. Para descartar el prefijo `'0b'` puede ejecutarse `bin(n).replace('0b','')`. No es necesario especificar ni verificar estas funciones en el TP.
- Considerar usar la función `round(n,digits)` de Python para redondear valores de tipo `float`.[2]
- Puede suponerse que el usuario siempre invocará las funciones de manera correcta. Es decir, si hay errores de tipo o valor de los argumentos provistos, no se espera comportamiento alguno del programa (podría colgarse o terminar en un error, por ejemplo).
- **Forma de evaluación:** Se evalúan tres aspectos de la entrega: (1) el *código*: no sólo su correcto funcionamiento, sino también que se sigan las buenas prácticas de programación vistas desde la primera clase: uso adecuado de tipos y estructuras de control, claridad en los nombres de variables y funciones auxiliares, comentarios, reutilización de funciones, etc.; (2) las *especificaciones* de las funciones; y (3) la *verificación* de los programas entregados, mediante testing de unidad, demostraciones de terminación y correctitud de ciclos, según corresponda.

[2] La documentación de referencia se encuentra en `https://docs.python.org/3/library/functions.html#round`.