Fundamentos de Algoritmos – Trabajo Práctico 1

Master in Management + Analytics, Escuela de Negocios, UTDT

Primer semestre 2020

Observaciones generales:

- El trabajo se debe realizar en grupos de **tres personas**.
- El código fuente debe estar bien comentado. Cualquier aclaración adicional que se considere necesaria debe ser incluida como comentarios en el código fuente.
- El archivo con el código fuente debe llamarse `tp1.py` y debe subirse a la *Entrega del TP1* en la página de la materia en el campus virtual.
- El programa entregado debe correr correctamente en `Python3`.
- La fecha límite de entrega es el **domingo 19 de abril a las 23:55**. Los TPs entregados fuera de término serán aceptados, pero la demora incidirá negativamente en la nota.

Números Primos

El estudio de los números primos es una parte importante de la teoría de números, la rama de la matemática que comprende el estudio de los números enteros. Los números primos están presentes en algunas conjeturas centenarias tales como la hipótesis de Riemann y la conjetura de Goldbach. Durante mucho tiempo se consideró que la aplicación de los números primos era muy limitada fuera de la matemática pura. Sin embargo, el desarrollo de la criptografía durante la década del '70 ha despertado un renovado interés sobre estos números: gran parte de los sistemas modernos de seguridad informática están basados en el hecho de que no se conoce un método eficiente para descomponer un número entero en sus factores primos. El presente trabajo tiene como objetivo la implementación de un programa que brinde funcionalidad de interés para quienes trabajan con estos números.

A continuación se enuncian algunas definiciones y resultados relevantes para el presente trabajo.

Definición 1. Un número $n \in \mathbb{N}$ es *primo* si y sólo si tiene sólo dos divisores positivos distintos: 1 y n.

Teorema 1. Existen infinitos números primos.

Teorema 2. Todo $n \in \mathbb{N}$ se puede representar de forma única como producto de factores primos.

Se pide implementar en Python funciones para:

- dado $n \in \mathbb{N}$, determinar si n es primo;
- dado $i \in \mathbb{N}$, obtener el i-ésimo número primo;
- dado $n \in \mathbb{N}$, obtener la cantidad de números primos menores o iguales a n;
- dado $n \in \mathbb{N}$, obtener la cantidad de divisores primos de n;
- dados $n, i \in \mathbb{N}$, obtener el i-ésimo divisor primo de n;
- dado $n \in \mathbb{N}$, obtener la suma de los primeros n números primos.

Las funciones deben tener el siguiente encabezado:

```
def esPrimo(n):
def iesimoPrimo(i):
def cantidadPrimosMenoresOIguales(n):
def cantidadDivisoresPrimos(n):
def iesimoDivisorPrimo(n, i):
def sumaPrimerosPrimos(n):
```

El programa presentado deberá poder ejecutarse desde la línea de comandos y recibirá como argumentos el nombre de la función que se desea ejecutar (por ejemplo, "`esPrimo`") más los argumentos necesarios para dicha función. A continuación se ilustran usos del programa con el resultado esperado:

Comando a ejecutar por línea de comandos	Resultado impreso por pantalla
`python tp1.py esPrimo 2`	`sí`
`python tp1.py esPrimo 6`	`no`
`python tp1.py iesimoPrimo 1`	`2`
`python tp1.py iesimoPrimo 5`	`11`
`python tp1.py cantidadPrimosMenoresOIguales 6`	`3`
`python tp1.py cantidadPrimosMenoresOIguales 7`	`4`
`python tp1.py cantidadDivisoresPrimos 6`	`2`
`python tp1.py cantidadDivisoresPrimos 7`	`1`
`python tp1.py iesimoDivisorPrimo 10 1`	`2`
`python tp1.py iesimoDivisorPrimo 10 2`	`5`
`python tp1.py iesimoDivisorPrimo 10 3`	`10 no tiene 3 divisores primos`
`python tp1.py sumaPrimerosPrimos 1`	`2`
`python tp1.py sumaPrimerosPrimos 3`	`10`
`python tp1.py sumaPrimerosPrimos 5`	`28`

Observaciones:

- Definir todas las funciones auxiliares que se consideren necesarias.
- Sólo está permitido importar la biblioteca `sys`, para el manejo de los argumentos del programa principal (`sys.argv`, por ejemplo).
- La función `print()` solamente puede ser invocada desde el cuerpo principal del código. Es decir, no puede usarse dentro de las funciones listadas arriba, ni dentro de una función auxiliar.
- Puede suponerse que el usuario siempre invocará al programa de manera correcta. Es decir, si hay errores en la cantidad, tipo o valor de los argumentos de entrada, no se espera comportamiento alguno del programa (podría colgarse o morir, por ejemplo).