The complexity zoo

Definición 6.1. Un alfabeto es un conjunto finito $\mathcal{A}$ de al menos dos elementos (llamados símbolos o letras). Denotamos $\mathcal{A}^n$ al conjunto de palabras de longitud exactamente n y $\mathcal{A}^*$ al conjunto de todas las palabras de longitud finita (incluyendo la de longitud cero). Dada una palabra $w \in \mathcal{A}^*$, denotamos por $|w|$ a su longitud.

Habitualmente se consideran el alfabeto de los bits $\mathcal{A} = \{0, 1\}$ o el de los bytes $\mathcal{A} = \{0, 1, \ldots, 255\}$. Sin embargo, en estas notas vamos a utilizar el alfabeto $\mathcal{A}$ de los caracteres UNICODE. Haciendo esto, identificaremos al conjunto $\mathcal{A}^*$ con los objetos de Python3 de tipo `str`.

Definición 6.2. Un lenguaje es un subconjunto $\mathcal{L} \subseteq \mathcal{A}^*$. El conjunto de todos los lenguajes es $\mathcal{P}(\mathcal{A}^*)$.

Por ejemplo, podemos considerar el lenguaje **Nat** de los números naturales escritos como cadenas de caracteres UNICODE (y sin ceros redundantes por delante). Se tiene que `"4675"` está en **Nat**, pero `"pepe"` y `"007"` no lo están. Otros lenguajes que nos van a interesar son el de los números primos **Prime** y el de los números compuestos **Composite**.

Definición 6.3. Un lenguaje $\mathcal{L}$ se dice de tipo P (resp. EXPTIME, DECIDIBLE) si es posible implementar la función

`pertenece_a_L(w)`

que toma un $w \in \mathcal{A}^*$ y devuelve un booleano, tal que para ciertos $C > 0$ y $k \geq 1$ se tiene:

1. Si $w \in \mathcal{L}$, la función devuelve `True`.
2. Si $w \notin \mathcal{L}$, la función devuelve `False`.
3. El tiempo de ejecución del algoritmo es $\leq C(1 + |w|)^k$ (resp. $\leq 2^{C(1+|w|)^k}$, finito).

Es bastante fácil ver que **Nat** es un lenguaje de tipo P. En efecto, el siguiente programa satisface la definición anterior con $k = 1$.

```
def pertenece_a_Nat(w):
    n = len(w)
    if n == 0:
        return False
    if w[0] == '0':
        return False
    for i in range(n):
        if not(w[i] in "0123456789"):
            return False
    return True
```

También es fácil comprobar que **Prime** es de tipo EXPTIME. En efecto, el siguiente programa comprueba si un $w \in \mathcal{A}^*$ está en **Prime** en $O(10^{|w|})$ operaciones aritméticas con enteros acotados por $10^{|w|}$. Cada una de estas operaciones se puede hacer en $O(|w|^2)$ operaciones binarias usando los algoritmos de la escuela.

```
def pertenece_a_Primes(w):
    if not pertenece_a_Nat(w):
        return False
    n = int(w)
    if n == 1:
        return False
    i = 2
    while i < n:
```

```
9           if n % i == 0:
10              return False
11          i += 1
12      return True
```

De la definición de puede deducir inmediatamente que

$$\mathrm{P} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{DECIDIBLE} \subsetneq \mathcal{P}(\mathcal{A}^*).$$

La última inclusión es estrica ya que DECIDIBLE es un conjunto numerable (porque una cantidad numerable de posibles funciones `pertenece_a_L(w)`) pero $\mathcal{P}(\mathcal{A}^*)$ es no numerable (por el argumento diagonal de Cantor).

Un teorema reciente, de Agrawal, Kayal y Saxena afirma que **Prime** es de tipo P, pero la demostración es bastante compleja. La complejidad del algoritmo es $O(|w|^{21/2+\varepsilon})$. Por supuesto, esto implica que

$$\mathbf{Composite} = \mathbf{Prime}^c \cap (\mathbf{Nat} \setminus \{\texttt{"1"}\})$$

es también de tipo P. En general, si $\mathcal{L}_1$ y $\mathcal{L}_2$ son lenguajes de tipo P (resp. EXPTIME, DECIDIBLE), entonces también $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{L}_1 \setminus \mathcal{L}_2$ y $\mathcal{L}_1^c$ son de tipo P (resp. EXPTIME, DECIDIBLE).

El lenguaje **Pair**, también llamado $\mathbf{Nat^2}$, consiste de las cadenas de caracteres `num1 + "," + num2`, donde `num1` y `num2` son elementos de **Nat**. Por ejemplo, la cadena `"234,7"` es un elemento de **Pair**, pero las cadenas `"pepe,8"`, `"55"`, `"8;3"` y `"0,0"` no lo son. De forma similar se define el lenguaje $\mathbf{Nat^k}$ para cualquier $k \geq 1$. También definimos el lenguaje $\mathbf{Nat}^*$ como la unión de todos los $\mathbf{Nat^k}$ con $k \geq 1$. Por abuso de notación, identificaremos $\mathbf{Nat^k}$ con el conjunto $\mathbb{N}^k$ y el conjunto $\mathbf{Nat}^*$ con el de las sucesión finitas de números naturales.

Problema 6.1. Demostrar que $\mathbf{Nat^k}$ es de tipo P para cualquier $k \geq 2$, implementado en Python3 la función que comprueba la pertenencia.

Problema 6.2. Sea $\mathcal{L} \subseteq \mathbf{Nat}$ el lenguaje de los números naturales que son potencias de primos. Demostrar, con ayuda del teorema AKS, que $\mathcal{L}$ es de tipo P.

Problema 6.3. Sea $\mathcal{L} \subseteq \mathbf{Pair}$ el lenguaje de los pares (g, p) tales que p es primo, $q = 2p+1$ es primo y g es una raíz primitiva módulo q. Demostrar, con ayuda del teorema AKS, que $\mathcal{L}$ es de tipo P.

Problema 6.4. Sea $\mathcal{L} \subseteq \mathbf{Pair}$ el lenguaje de los pares (g, p) tales que p es primo, $p-1$ se factoriza como producto de primos menores que 1000 y g es una raíz primitiva módulo p. Demostrar que $\mathcal{L}$ es de tipo P. ¿Como podría hacerse esto sin utilizar el teorema AKS?

Problema 6.5. Sea $\mathcal{L} \subseteq \mathbf{Nat^3}$ el lenguaje de las ternas (a, b, n) tales que $n \geq 3$ es potencia de un primo impar y la ecuación $x^2 + ax + b \equiv 0 \pmod{n}$ tiene solución. Probar que $\mathcal{L}$ es de tipo P.

Definición 6.4. Un lenguaje $\mathcal{L} \subseteq \mathcal{A}^*$ se dice de tipo NP (resp. NEXPTIME, NDECIDIBLE) si es posible implementar la función

`es_certificado_valido_L(w,c)`

que toma $w, c \in \mathcal{A}^*$ y devuelve un booleano, tal que para ciertos $C_1, C_2 > 0$ y $k_1, k_2 \geq 1$ se tiene:

1. Si $w \in \mathcal{L}$, existe un $c \in \mathcal{A}^*$ de longitud $\leq C_1(1 + |w|)^{k_1}$ (resp. $\leq 2^{C_1(1+|w|)^{k_1}}$, finita), tal que la función devuelve `True`.
2. Si $w \notin \mathcal{L}$, para cualquier $c \in \mathcal{A}^*$, la función devuelve `False`.
3. El tiempo de ejecución del algoritmo es $\leq C_2(1 + |w|)^{k_2}$ (resp. $\leq 2^{C_2(1+|w|)^{k_2}}$, finito).

En el apartado 1, se dice que c es un *certificado sucinto* de $w \in \mathcal{L}$.

Todo lenguaje de tipo P es también de tipo NP, ya que se podría utilizar la función `pertenece_a_L(w)` para implementar la función `es_certificado_valido_L(w,c)`, ignorando completamente el argumento c. Se dice que P $\subseteq$ NP. Del mismo modo se concluye que EXPTIME $\subseteq$ NEXPTIME y DECIDIBLE $\subseteq$ NDECIDIBLE.

Si $\mathcal{L}_1$ y $\mathcal{L}_2$ son de tipo NP, entonces $\mathcal{L}_1 \cup \mathcal{L}_2$ y $\mathcal{L}_1 \cap \mathcal{L}_2$ son también de tipo NP. Sin embargo, debido a la asimetría entre los apartados 1 y 2 de la defínción 6.4, no es claro si $\mathcal{L}_1^c \in NP$. Lo mismo es válido para NEXPTIME y NDECIDIBLE.

Definición 6.5. Un lenguaje $\mathcal{L} \subseteq \mathcal{A}^*$ se dice de tipo coNP (resp. coNEXPTIME, coNDEDICIBLE) si $\mathcal{L}^c$ es de tipo NP (resp. NEXPTIME, DECIDIBLE).

Veamos que el lenguaje **Composite** es de tipo NP sin utilizar el teorema AKS. Para cada w compuesto, su certificado sucinto podría ser un par $c = (c_1, c_2) \in \mathbf{Pair} \subseteq \mathcal{A}^*$ tal que $w = c_1 c_2$ y $c_1, c_2 \geq 2$. Es fácil construir una función (de complejidad polinomial) que compruebe si dados $w, c \in \mathcal{A}^*$ se cumplan todas esas condiciones. Evidentemente el certificado sucinto garantiza que w es compuesto y ningún primo tiene un certificado válido. Por supuesto, esto implica que **Prime** es de tipo coNP.

Es bastante más dificil demostrar que **Prime** es de tipo NP sin utilizar el teorema AKS. De hecho, salvo cuestiones de notación, eso es lo que hemos hecho en el problema de entregable #4. Se certifica la primalidad de w a través del teorema de Lucas.

La función `es_certificado_valido_L(w,c)` puede convertirse fácilmente en un test de pertenencia `pertenece_a_L(w)` probando con todos los posbiles $c \in \mathcal{A}^*$ de longitud $|c| \leq C_1(1 + |w|)^{k_1}$, pero lamentablemente eso no da un algoritmo polinomial. En el peor caso, es decir cuando $w \notin \mathcal{L}$, se tendrán que hacer $|\mathcal{A}|^{C_1^2(1+|w|)^{2k_1}}$ comprobaciones. De todos modos, este razonamiento prueba que NP $\subseteq$ EXPTIME. También se concluye que NDECIDIBLE = DECIDIBLE $\cap$ coNDECIDIBLE, ya que en las definiciones de estas clases de complejidad no se impone ningún límite sobre el tiempo de ejecución (salvo que este sea finito).

Problema 6.6. Sea $\mathcal{L} \subseteq \mathbf{Nat^4}$ el lenguaje de las 4-tuplas (a, b, n, c) tales que la ecuación $x^2 + ax + b \equiv 0 \pmod{n}$ tiene una solución $0 \leq x \leq c$. Demostrar que $\mathcal{L}$ es de tipo NP.

Problema 6.7. Sea $\mathcal{L} \subseteq \mathbf{Nat}$ el lenguaje de los números (compuestos) de Carmichael. Demostrar que $\mathcal{L} \in$ NP $\cap$ coNP. ¿Como puede hacerse sin utilizar el teorema AKS?

Problema 6.8. Sea $\mathcal{L} \subseteq \mathbf{Pair}$ el lenguaje de los pares (g, p) tales que $p \geq 2$ es primo y g es una raíz primitiva módulo p. Demostrar que $\mathcal{L} \in$ NP $\cap$ coNP.

Definición 6.6. Un lenguaje $\mathcal{L} \subseteq \mathcal{A}^*$ se dice de tipo RP si es posible implementar la función

`es_certificado_valido_L(w,c)`

que toma $w, c \in \mathcal{A}^*$ y devuelve un booleano, tal que para ciertos $C_1, C_2 > 0$ y $k_1, k_2 \geq 1$ se tiene:

1. Si $w \in \mathcal{L}$, entonces para al menos la mitad de los $c \in \mathcal{A}^*$ de longitud $\leq C_1(1 + |w|)^{k_1}$, la función devuelve `True`.
2. Si $w \notin \mathcal{L}$, para cualquier $c \in \mathcal{A}^*$, la función devuelve `False`.
3. El tiempo de ejecución del algoritmo es $\leq C_2(1 + |w|)^{k_2}$.

En el apartado 1, se dice que c es un *certificado sucinto* de $w \in \mathcal{L}$.

Comparando con la definición de la clase NP, se puede ver que la única diferencia está en que para RP se pide que al menos la mitad de los posibles $c \in \mathcal{A}^*$ de longitud $|c| \leq C_1(1+|w|)^{k_1}$ certifiquen la pertenecia, cuando en NP solo se pide la existencia de (al menos) un certificado. Claramente se tiene que P $\subseteq$ RP $\subseteq$ NP.

Si bien es posible transformar la función `es_certificado_valido_L(W,b)` de un lenguaje $\mathcal{L}$ de tipo RP en un test determinista de pertenencia `pertenece_a_L(w)`, probando con todos los posibles c de longitud acotada por $C_1(1+|w|)^{k_1}$, esto nos daría un algoritmo exponencial en $|w|$. Una idea mejor es implementar un test *probabilista* de pertenencia a $\mathcal{L}$. Se eligen $c_1, c_2, \ldots, c_k$ uniformememnte al azar y luego se comprueba si algún certificado c_i valida a w. En caso afirmativo, hemos demostrado que $w \in \mathcal{L}$. Por otra parte, si $w \in \mathcal{L}$, la probabilidad de que ninguno de los c_i valide a w es $\leq (1/2)^k$.

La proposición 5.7 del tema #5 demuestra que **Composite** es de tipo RP. El certificado sucinto de la no primalidad de un entero $n \geq 3$ impar es un $a \in \mathbb{Z}$ tal que $\gcd(a,n) = 1$ y $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$. Al menos la mitad de los posibles a entre 1 y n satisfacen esta condición. El algoritmo de pertenecia que se obtiene es el de Solovay-Strassen.

Una definición equivalente de la clase RP es la siguiente:

Definición 6.7. Un lenguaje $\mathcal{L} \subseteq \text{Alf}^*$ se dice de tipo RP si es posible implementar un algoritmo probabilista

`pertenece_a_L(w)`

que toma un $w \in \mathcal{A}^*$ y devuelve un booleano, tal que para ciertos $C > 0$ y $k \geq 1$ se tiene:

1. Si $w \in \mathcal{L}$, devuelve `True` con probabilidad $\geq \frac{1}{2}$.
2. Si $w \notin \mathcal{L}$, siempre devuelve `False`.
3. El tiempo de ejecución es $\leq C(1+|w|)^k$.

Ya hemos demostrado que la primer definición implica la segunda. Para ver la otra podríamos tomar $C_1 = C_2 = C$ y $k_1 = k_2 = k$ y sustituir cada invocación a la función random por leer de la cadena c. Haciendo esto, mas de la mitad de los posibles c validarán un w que pertenezca a $\mathcal{L}$.

Debe notarse que si el algoritmo probabilista de la definición 6.7 devuelve `True`, eso demuestra que $w \in \mathcal{L}$. Por otra parte, si devuelve `False`, eso significa que o bien $w \notin \mathcal{L}$ o que $w \in \mathcal{L}$ y tuvimos mala suerte con el azar. Repitiendo el test muchas veces se puede reducir la probabilidad de esto último tanto como se quiera.

Definición 6.8. Un lenguaje $\mathcal{L} \subseteq \mathcal{A}^*$ se dice de tipo coRP si $\mathcal{L}^c$ es de tipo RP. La intersección RP $\cap$ coRP se llama ZPP.

Problema 6.9. Demostrar que en la definción 6.7 se puede reemplazar $\frac{1}{2}$ por cualquier $\varepsilon \in (0,1)$.

Problema 6.10. Demostrar que el lenguaje de los números (compuestos) de Carmichael es coRP.

Problema 6.11. Sea $\mathcal{L} \subseteq \mathcal{A}^*$ un lenguaje. Demostrar que son equivalentes:

1. $\mathcal{L} \in$ ZPP.
2. Se puede implementar la función `pertenece_a_L(w)` con un algoritmo probabilista, que toma un $w \in \mathcal{A}^*$ y devuelve un booleano o `"no lo se"`, de modo que:
   - Si devuelve `True`, entonces $w \in \mathcal{L}$.
   - Si devuelve `False`, entonces $w \notin \mathcal{L}$.
   - La probabilidad de que devuelve `"no lo se"` es $\leq \frac{1}{2}$.
   - El tiempo de ejecución del algoritmo es $\leq C(1+|w|)^k$ para ciertos $C > 0$ y $k \geq 1$.

3. Se puede implementar la función `pertenece_a_L(w)` con un algoritmo probabilista, que toma un $w \in \mathcal{A}^*$ y devuelve un booleano, de modo que:
   - Si devuelve `True`, entonces $w \in \mathcal{L}$.
   - Si devuelve `False`, entonces $w \notin \mathcal{L}$.
   - El tiempo de ejecución del algoritmo es una variable aleatoria con esperanza $\leq C(1+|w|)^k$ para ciertos $C > 0$ y $k \geq 1$.

Definición 6.9. Una algoritmo probabilista como el del apartado 2 o el 3 del problema 6.11 se dice de tipo Las Vegas. Estos algoritmos siempre dicen la verdad.

Definición 6.10. Un lenguaje $\mathcal{L} \subseteq \mathcal{A}^*$ se dice de tipo BPP si es posible implementar la función `es_certificado_valido_L(w,c)` con un algoritmo determinista, que toma $w, c \in \mathcal{A}^*$ y devuelve un booleano, tal que existen $C_1, C_2 > 0$ y $k_1, k_2 \geq 1$, de modo que:

- Si $w \in \mathcal{L}$, devuelve `True` para al menos $2/3$ de los $c \in \mathcal{A}^*$ de longitud $\leq C_1(1+|w|)^{k_1}$.
- Si $w \notin \mathcal{L}$, devuelve `False` para al menos $2/3$ de los $c \in \mathcal{A}^*$ de longitud $\leq C_1(1+|w|)^{k_1}$.
- El tiempo de ejecución es $\leq C_2(1+|w|)^{k_2}$.

Si interpretamos al certificado suciento c como una cadena aleatoria, llegamos a la siguiente definición equivalente de la clase BPP:

Definición 6.11. Sea $\mathcal{L} \subseteq \mathcal{A}^*$ un lenguaje. Se dice que $\mathcal{L}$ es de tipo BPP si es posible implementar la función `pertenece_a_L(w)` con un algoritmo probabilista, que toma $w \in \mathcal{A}^*$ y devuelve un booleano, de modo que:

1. prob(devuelve `True` $\mid w \in \mathcal{L}) \geq \frac{2}{3}$.
2. prob(devuelve `False` $\mid w \notin \mathcal{L}) \geq \frac{2}{3}$.
3. El tiempo de ejecución es $\leq C(1+|w|)^k$ para ciertos $C > 0$ y $k \geq 1$.

Tal como hicimos antes, eligiendo $c_1, c_2, \ldots, c_r$ al azar de longitud $\leq C_1(1+|w|)^k$, y adoptando la decisión de la mayoría de `es_certificado_valido_L(w,c_i)`, obtenemos un algoritmo probabilista para la pertenencia a $\mathcal{L}$ con una probabilidad de error tan pequeña como necesitemos.

Problema 6.12. Demostrar que la definiciones 6.10 y 6.11 son equivalentes y que en cualquiera de ellas es posible reemplazar el $2/3$ por cualquier $\varepsilon \in (1/2, 1)$.

Un algoritmo probabilista como el de la definición 6.11 no siempre devuelve la verdad, pero la probabilidad de error puede reducirse tanto como se quiera. El tiempo de ejecución de estos algoritmos es siempre polinomial en $|w|$. Estos algoritmos de dicen de tipo Montecarlo.

Problema 6.13. Sea $\mathcal{L} \subseteq \mathbf{Nat}^4$ el lenguaje de las 4-tuplas (a, b, n, c) tales que $x^2 + ax + b \equiv 0 \pmod{n}$ tiene solución $x \in [0, c]$. Demostrar que si $\mathcal{L} \in \text{BPP}$, entonces $\mathcal{L} \in \text{RP}$.
Hint: Utilizar el método de la bisección en la variable c.