Examen de Algorítmica y Complejidad (Plan 2014) — **17 de diciembre de 2018**

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema. La compañía "*Geneticáte SA*" está especializada en la síntesis de secuencias de ADN[1]. La compañía sintetiza una secuencia de ADN a partir de otra a través de las siguientes operaciones (cada una de ellas con un coste asociado):

- Introduciendo un nucleótido. Esta modificación tiene un coste a.
- Borrando un nucleótido. Esta modificación tiene un coste b.
- Mutando un nucleótido de tipo i por otro de tipo j. Esta modificación tiene un coste m_{ij}. Por ejemplo m_{AT} representa el coste de mutar una **A**denina por una **T**imina

La compañía nos pide diseñar un algoritmo que calcule el **mínimo** coste que supone sintetizar una secuencia de ADN a partir de otra (teniendo en cuenta los valores $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{m_{ij}}$).

Ejemplo: Considerando los costes son $a = 1$; $b = 2$.

m_{ij}	A	G	C	T
A	0	1	2	2
G	2	0	1	2
C	2	1	0	3
T	3	1	2	0

El coste mínimo que supone sintetizar la secuencia AGGGTGCA a partir de AGCGTGCGA es **3**:

AG**C**GTGCGA → AG**G**GTGCGA (coste +1 por mutar C en G)
AGGGTGC**G**A → AGGGTGCA (coste +2 por borrar un nucleótido)

Para realizar este cálculo vamos a implementar un algoritmo basado en **programación dinámica** con la siguiente cabecera:

```
int coste(int[] adn1, int[] adn2, int a, int b, int[][] m)
```

donde:

- Cada secuencia de ADN está codificada por un vector de números entre 0 y 3 (representando los cuatro tipos de nucleótidos que hay en el ADN: 0-**A**denina; 1-**G**uanina; 2-**C**itosina; 3-**T**imina). Por ejemplo la secuencia AGCGTGCGA está representado por este vector:

0	**1**	**2**	**1**	**3**	**1**	**2**	**1**	**0**

- `int a, int b` representan respectivamente los costes de añadir y borrar un nucleótido.
- `int[][] m` es una matriz 4x4 representando los costes de mutar un tipo de nucleótido por otro.

[1] Una secuencia de ADN es una sucesión de los 4 tipos posibles de nucleótidos: **A**denina, **G**uanina, **C**itosina, **T**imina. Un ejemplo de secuencia de ADN sería el siguiente AGCGTGCGA.

a) Define la **entrada**, la **salida** y la **semántica** de la función sobre la que estará basado el algoritmo de programación dinámica.

C(*i*,*j*): coste de transformar la subcadena de *adn1*[0…*i*-1] en la subcadena *adn2*[0…*j*-1].
Entrada: i≥0,j≥0 (2 enteros). Los valores i=0, j=0 representan respectivamente las subcadenas de adn1 y adn2 vacías
Salida: valor entero (indica el coste de la transformación)

b) Expresa recursivamente la función anterior.

Casos bases: $C(i,0) = i \cdot b$; $C(0, j) = j \cdot a$
Caso recursivo: Para $i>0$; $j>0$

$$C(i, j)=\min\{m_{adn[i-1],adn2[j-1]} + C(i-1,j-1),\ a + C(i,j-1),\ b + C(i-1,j)\}$$

c) Basándote en los anteriores apartados implementa un algoritmo de programación dinámica que tenga complejidad en tiempo $O(N \cdot M)$ y en memoria $O(N)$ (donde N es el tamaño de la secuencia de ADN más pequeño y M el tamaño de la secuencia de ADN más grande).[2]

```
int coste(int[] adn1, int[] adn2, int a, int b, int[][] m) {
   int long1= adn1.length;
   int long2= adn2.length;
   if (long2<long1) {
     int[][] c = new int[2][long2+1];
     for (int j=0; j<=long2; j++)
        c[0][j]=j*a;
     for (int i=1; i<=long1; i++){
        c[i%2][0]=i*b;
        for (int j=1; j<=long2; j++)
           c[i%2][j]= Math.min(m[adn1[i-1]][adn2[j-1]]+c[(i-1)%2][j-1],
                 Math.min(a+c[i%2][j-1],b+c[(i-1)%2][j]));
     }
     return c[long1%2][long2];
   }
   else {
     int[][] c= new int[long1+1][2];
     for (int i=0; i<=long1; i++)
        c[i][0]=i*b;
     for (int j=1; j<=long2; j++) {
        c[0][j%2]=j*a;
        for (int i=1; i<=long1; i++)
            c[i][j%2]=Math.min(m[adn1[i-1]][adn2[j-1]]+c[i-1][(j-1)%2],
                    Math.min(a+c[i][(j-1)%2],b+c[i-1][j%2]));
     }
     return c[long1][long2%2];
   }
}
```

[2] No cumplir con los requisitos de complejidad pedidos conlleva **una puntuación de 0** en este apartado.