Ceros y doses

Consideremos la lista ordenada de todos los números enteros entre 0 y (3^n)-1 expresados en base 3. Ahora, dividimos esta lista en bloques de números consecutivos de manera que, en cada bloque, o bien todos los números tienen algún uno (*1*) en su expresión, o bien ninguno de ellos tiene unos en su expresión. Elegimos los bloques más grandes posibles, de manera que están determinados de manera única. Tomemos, por ejemplo, n=2. En este caso, los bloques son *(00),(01),(02),(10,11,12),(20),(21),(22)*. Te pedimos que imprimas los tamaños, en orden, de todos estos bloques. En este caso, sería *1 1 1 3 1 1 1*.

Estrategia de la solución

La única dificultad que tenía este problema era darse cuenta del patrón que siguen las expresiones en base 3 de los primeros 3^n números naturales. En el caso n=0, hay un único número (el 0), que no tiene unos en su expresión. En el caso n=1, tenemos los números 0, 1 y 2, de los cuales solamente el 1 tiene unos en su expresión. Para n=2, tenemos los números 0, 1, 2, 10, 11, 12, 20, 21, 22. Podemos observar que el primer tercio de los números es el caso anterior, que el segundo tercio es el caso anterior añadiendo un uno al principio de la expresión (lo que implica que tienen unos en su expresión), y que el tercer tercio de los números es el caso anterior añadiendo un dos al principio de la expresión. Así pues, podemos crear una función recursiva que, dada un n, se llame a si misma para n-1, imprima 3^(n-1) y se vuelva a llamar a si misma con n-1. El caso base es n=0, para el que basta imprimir un 1.

Solución en Python

```
# funcion que dibuja los tamanos de los bloques
def dibuja_bloques(n):
        if n == 0:
        print(1)
        return

        dibuja_bloques(n-1)

        print(3**(n-1))

        dibuja_bloques(n-1)

# leemos una linea del input
s = input()

# si esta vacia, hemos acabado
while s != '':

        # sino, s contine un numero del input
        n = int(s)

        dibuixa_blocks(n)
        print('');
```

```
        # leemos la siguiente linea del input
        s = input()
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>

using namespace std;

void fun(int x)
{
        if (x == 0) {
        cout << 1 << endl;
        return;
        }
        fun(x - 1);
        int m = 1;
        for (int i = 0; i < x - 1; ++i)
        m = m * 3;
        cout << m << endl;
        fun(x - 1);
        return;
}

int main()
{
        int n;
        while (cin >> n) {
        fun(n);
        cout << endl;
        }
}
```

Autor de la solución: Jordi Castellví (Miembro del comité organizador)

Eleuteria y los palíndromos no triviales

Eleuteria estaba rebuscando en su cajón y se ha encontrado una palabra especialmente fea. A Eleuteria no le gustan nada los palíndromos no triviales - palabras de dos o más letras que se leen igual del derecho y del revés. Por este motivo, si la palabra contiene alguno, elegirá uno al azar y lo eliminará, reduciendo el tamaño de la palabra. Eleuteria aplica este procedimiento sucesivamente hasta que ya no queden palíndromos no triviales en su palabra. Determina si es posible que Eleuteria elimine todas las letras de la palabra.

Estrategia de la solución

La solución esperada para este problema es una búsqueda exhaustiva dónde exploramos todas las formas posibles de eliminar sucesivamente un palíndromo de la palabra para ver si en algún caso conseguimos eliminar la palabra entera. Para hacer esto, primero podíamos observar que basta con eliminar solamente palíndromos de longitud 2 o 3, puesto que dado un palíndromo de longitud arbitraria, necesariamente contiene un palíndromo (en el medio) de longitud 2 o 3 tal que, cuando lo eliminamos, lo que sobra sigue siendo un palíndromo. De esta forma, podemos escribir una función recursiva que dada una palabra determine si se pueden eliminar todas las letras. Si la palabra que recibe la función es vacía, entonces la respuesta es afirmativa. De lo contrario, la función busca todos los palíndromos de dos y tres letras contenidos en la palabra y para cada uno se llama recursivamente con la palabra resultante de eliminar el palíndromo en cuestión. Si alguna de estas llamadas recursivas responde afirmativamente, significa que se pueden eliminar todas las letras y la función emitirá una respuesta afirmativa. De no ser así, la respuesta será negativa. Cabe destacar que cada llamada a la función puede tener un coste en tiempo lineal en función de la longitud de la palabra si la programamos de forma inteligente, a diferencia de la solución ingenua, que tendría un coste cuadrático. Por último, si bien es cierto que podríamos optimizar el código guardando las palabras que ya sabemos que tienen una respuesta negativa para no resolverlas más veces de las necesarias, el objetivo del problema era practicar el backtracking y no era necesario implementar esta mejora.

Solución en Python

```python
# funcion que determina si a la string s
# se puede limpiar de palindromos
def eliminar_palindromos(s):

    if len(s) == 0: return True
    if len(s) == 1: return False

    # t es la substring s[0:len(s)-3]
    t = s[0:-2]

    for i in range(len(s) - 1):
        if (s[-1-i] == s[-2-i] and eliminar_palindromos(t)):
                return True
        elif (i < len(s)-2):
                t[-1-i] = s[-1-i]
```

```
    if len(s) == 2: return False

    # t es la substring s[0:len(s)-4]
    t = s[0:-3]

    for i in range(len(s)-2):
        if (s[-1-i] == s[-3-i] and eliminar_palindromos(t)):
               return True
        elif (i < len(s) - 3):
               t[-1-i] = s[-1-i]

    return False

s = input()
while s != '':

    # en python las strings no se pueden modificar
    # transformamos s en lista para poder cambiar sus elementos
    s = list(s)

    if eliminar_palindromos(s):
        print('SI')
    else:
        print('NO')

    s = input()
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>

using namespace std;

bool f(string s) {
       if (s.size() == 0)
       return true;
       if (s.size() == 1)
       return false;

       string t = s.substr(0, s.size() - 2);

       for (int i = 0; i < s.size() - 1; ++i) {
       if (s[s.size() - 1 - i] == s[s.size() - 2 - i] and f(t))
              return true;
       if (i < s.size() - 2)
              t[t.size() - 1 - i] = s[s.size() - 1 - i];
       }

       if (s.size() == 2)
       return false;
```

```
        t = s.substr(0, s.size() - 3);

        for (int i = 0; i < s.size() - 2; ++i) {
        if (s[s.size() - 1 - i] == s[s.size() - 3 - i] and f(t))
                return true;
        if (i < s.size() - 3)
                t[t.size() - 1 - i] = s[s.size() - 1 - i];
        }

        return false;
}

int main() {
        string s;
        while (cin >> s) {
        if (f(s))
                cout << "SI" << endl;
        else
                cout << "NO" << endl;
        }
        return 0;
}
```

Autor de la solución: Jordi Castellví (Miembro del comité organizador)

Secuencias formidables

Tras horas de trabajo, Rodofredo ha terminado de confeccionar su lista de secuencias formidables de *n* ceros y *m* unos. Una secuencia formidable de ceros y unos es un conjunto ordenado de ceros y unos tal que el tamaño de las subsecuencias de ceros consecutivos es estrictamente decreciente y el tamaño de las secuencias de unos consecutivos es estrictamente creciente. Por ejemplo, la secuencia *(000100110)* es formidable porque 3 > 2 > 1 (para los ceros) y 1 < 2 (para los unos). En cambio, la secuencia *(100111000)* no es formidable porque primero hay dos ceros consecutivos y luego tres.

Rodofredo decide que su maravillosa creación es digna de ser compartida y, despúes de ordenar las secuencias en orden lexicográfico, se dispone a mandar el resultado a su amiga Clotilde. Sin embargo, se percata de que su conexión a internet está muy ralentizada, puesto que se está descargando el Fortnite, de modo que no puede enviar una lista tan larga. Así pues, decide enviar una sola secuencia formidable, la i-ésima, y el número total de secuencias formidables de su lista.

Te pedimos que escribas la información que recibirá Clotilde.

Estrategia de la solución

La solución esperada para este problema consistía en generar todas las secuencias formidables con n ceros y m unos e imprimir solamente la i-ésima. Para ello, podemos escribir una función recursiva que genere todas las secuencias formidables con un número dado de ceros y unos sabiendo cual es máximo número de ceros y el mínimo número de unos que puede escribir. La implementación de esta función es la natural y no tiene ninguna complicación. Existe una optimización y consiste en parar la búsqueda exhaustiva cuando sabemos que no podremos completar la secuencia formidable. Esto ocurre cuando el máximo número de ceros que podemos llegar a escribir de un punto en adelante no es suficiente para llegar a n. Esta optimización mejora el tiempo de ejecución pero es poco significativo en los casos pequeños, así que no es necesaria para pasar los casos de prueba.

Solución en C++

```
#include <iostream>
#include <vector>
#include <math.h>

using namespace std;

int cont;
vector<int> v;

void rec(int p, int i, int s, int t, int n, int m, int x) {
	if (i == v.size()) {
	++cont;
	if (x == cont) {
```

```
                cout << "(" << v[0];
                for (int j = 1; j < v.size(); ++j)
                cout << ", " << v[j];
                cout << ")\n";
        }
        return;
        }
        if (p) {
        if (m <= t)
                return;
        for (int j = 0; j < t; ++j)
                v[i + j] = 1;
        for (int j = 0; j < m - t; ++j) {
                v[i + t + j] = 1;
                rec(!p, i + t + j + 1, s, t + j + 1, n, m - (t + j + 1), x);
        }
        }
        else {

        //##### posible optimización #####
        int k = (sqrt((2 * t + 1) * (2 * t + 1) + 8 * m) - 2 * t - 1) / 2;
        if (n > (k + 1) * s - (k + 1) * (k + 2) / 2 and not(k >= s - 2 and s * (s - 1) / 2
>= n))
                return;
        //################################

        for (int j = 0; j < min(n, s - 1); ++j)
                v[i + j] = 0;
        for (int j = min(n, s - 1) - 1; j >= 0; --j)
                rec(!p, i + j + 1, j + 1, t, n - (j + 1), m, x);
        }
        return;
}

int main() {
        int n, m, x;
        while (cin >> n >> m >> x) {
        v = vector<int>(n + m);
        cont = 0;
        rec(0, 0, 1e6, 0, n, m, x);
        rec(1, 0, 1e6, 0, n, m, x);
        cout << cont << "\n----------\n";
        }
}
```

Autor de la solución: Jordi Castellví (Miembro del comité organizador)