Control 6

Contando números

El objetivo de este control es familiarizarse con las técnicas de generalización e inmersión en el diseño de algoritmos recursivos, así como con la determinación de la complejidad de dichos algoritmos mediante la definición y resolución de ecuaciones de recurrencia.

El problema

Debemos diseñar un algoritmo recursivo que cuente la cantidad de números naturales menores que un *n* dado ($n \geq 0$), que no tengan dígitos repetidos.

Tras realizar un primer análisis del problema, hemos llegado a las siguientes conclusiones:

- Si n = 0, o $n \geq 10000000000$ (escrito en C++, `10000000000LL`: el sufijo `LL` indica que es un entero de tipo `long long`, representado con, al menos 64 bits), podemos determinar directamente el resultado:
  - Cuando n=0, el resultado es 0 (no hay ningún número natural menor que 0)
  - Cuando $n \geq 10000000000$, el resultado será el número de todos los naturales que no tienen dígitos repetidos. Aplicando un poco de combinatoria llegamos a que dicha cantidad es 8877691 (`8877691LL`, en C++).
- En otro caso, conviene expresar el resultado como A + B + 1, donde (i) A es la cantidad de números positivos sin dígitos repetidos y con *menos* dígitos que *n*; y (ii) B es la cantidad de números positivos menores que *n*, sin dígitos repetidos, y con exactamente el mismo número de dígitos que *n*. La suma de 1 es por el 0, que no está contabilizado ni en A ni en B. El cálculo de A y B constituye el núcleo del algoritmo recursivo:
  - El cálculo de la cantidad A de números positivos sin dígitos repetidos que tienen *menos* dígitos que *n* se lleva a cabo teniendo en cuenta que:
    - Hay exactamente $C_1 = 9$ de estos números que tienen exactamente un dígito.
    - Si C_k es la cantidad de tales números que tienen exactamente *k* dígitos, entonces $C_{k+1} = C_k \times (10 - k)$ es la cantidad de los que tienen exactamente *k*+1 dígitos.
    - Por tanto, $A = C_1 + C_2 + \ldots + C_{D-1}$, con D el número de dígitos de *n*.
  - El cálculo de la cantidad B de números positivos menores que *n*, sin dígitos repetidos, y que tienen el mismo número de dígitos que *n* se realiza considerando que:
    - Si *n* es un único dígito, B = *n*-1
    - Si *n* es de la forma *n*'**d** (*n*' otro número, **d** un dígito), y B' es la cantidad para *n*', entonces $B = B_0 + B_1$ donde:
      - $B_0 = B' \times (10 - D')$, donde D' es el número de dígitos de *n*'
      - B_1 es la cantidad de números de la forma *n*'**d'**, tal que **d'** < **d** y además **d'** no aparece en *n*'.

Tu labor consiste en (i) diseñar una generalización que permita computar recursivamente, y de manera simultánea, los valores A y B descritos anteriormente para *n*; (ii) diseñar el algoritmo pedido como una inmersión de la generalización citada. Observa que, para realizar su tarea de manera eficiente, aparte de devolver los valores A y B, la generalización podrá incluir todos aquellos argumentos adicionales que consideres oportuno.

Trabajo a realizar

Debe diseñarse el algoritmo recursivo pedido, completando los apartados indicados entre comentarios en el archivo `control6.cpp` que se proporciona como apoyo. Debe implementarse, además, el algoritmo. El punto de entrada al mismo será la función `num_sinrepetidos_menoresque`, donde se definirá el algoritmo como una inmersión de la generalización definida por la función `cuenta_sinrepetidos_menoresque`. Aparte de llevar a cabo el diseño del algoritmo, e implementar el mismo, deberás determinar justificadamente su complejidad, definiendo y resolviendo las recurrencias necesarias (la resolución se llevará a cabo utilizando los patrones de recurrencias genéricas discutidos en clase).

Programa de prueba

Se proporcionan dos versiones alternativas (puede elegirse una u otra comentando o descomentando la definición `#define DOM_JUDGE` que se encuentra al comienzo del archivo):

- Si DOM_JUDGE está definido, el programa lee por la entrada estándar enteros representables como valores del tipo `long long` (enteros que utilizan, como mínimo, 64 bits). Todos los enteros leídos serán no negativos, excepto el último, que será -1 y marcará el final de los casos

de prueba. Cada vez que lee un caso de prueba, el programa invoca a `num_sinrepetidos_menoresque` e imprime el resultado. A continuación, se muestra algunos ejemplos de entrada / salida:

Entrada	Salida
8 96 5698 193456785320 -1	8 88 3090 8877691

- Si DOM_JUDGE no está definido, el programa genera aleatoriamente 1000 números y compara el resultado del algoritmo con una implementación *naif* del mismo. Este modo de ejecución es útil para someter al algoritmo a una prueba de estrés. Si se genera un número para el cuál la implementación recursiva y la implementación *naif* discrepan, el programa termina y muestra el valor para el que el algoritmo falla. Para activar este modo basta comentar la línea `#define DOM_JUDGE`

El archivo completo debe entregarse a través del juez en línea de la asignatura.

Importante:

- Únicamente se evaluarán aquellas entregas que superen satisfactoriamente los casos de prueba del juez.
- No modificar el código proporcionado. Únicamente debe implementarse las funciones `num_sinrepetidos_menoresque` y `cuenta_sinrepetidos_menoresque`.
- **No se corregirá ninguna entrega en la que no se hayan incluido los nombres de los miembros del grupo que han realizado el trabajo en el comentario que se incluye al comienzo del archivo.**