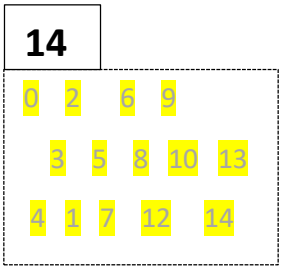
En el control 6 pedimos diseñar un algoritmo que encuentre la cantidad de *números singulares* menores que uno dado *n* ($n \geq 0$). Para ello, de acuerdo con el enunciado, podemos ver que un número es *singular* cuando su dígito más significativo no vuelve a aparecer en el resto del número. Por ejemplo, 8776 es singular, ya que 8 (el dígito más significativo de 8776) no aparece en 776. Pero 8778 no es singular, ya que 8 (el dígito más significativo de 8778) sí vuelve a aparecer en 778.

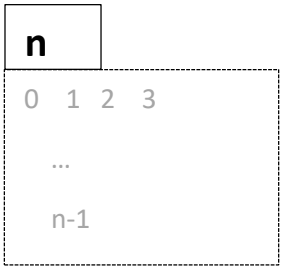
La clave para resolver este problema (y otros similares) **es pensar en la cantidad de números pedida, no únicamente como un valor, sino como la representación de la caja de todos los números pedidos**.

Por ejemplo, si *n*=15, sabemos que el resultado es 14. Pero a este 14, en nuestra *cabeza*, no le tenemos que ver únicamente como una cantidad, sino como una cantidad que resume una propiedad de una caja de números: la caja formada por todos los números singulares menores que 15:



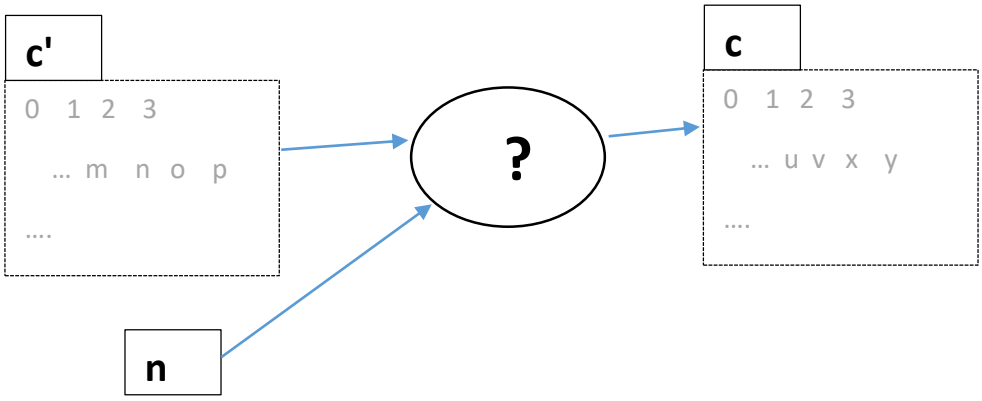

Por supuesto que, tomando *n*=15 como entrada, nuestro algoritmo devolverá 14 como resultado. Pero, la visión mental que tenemos que tener es la del 14 *pegado* a la caja de los números que, en este caso, está contabilizando. Dicha visión (la cantidad junto a la caja de números a partir de la que se obtiene) es fundamental para diseñar adecuadamente el algoritmo.

El caso más simple (caso base) es cuando *n* tiene un único dígito (es decir, n <= 9). En este caso, los números singulares más pequeños que *n* serán, exactamente, todos los números mayores o iguales que 0 y menores que *n* (porque los números de un dígito siempre son singulares, ya que no hay resto en el que pueda aparecer dicho dígito, que es el más significativo de dicho número). Por tanto, el resultado es:




Supongamos, ahora, que *n* tiene más de un dígito (es decir, n>9; caso recursivo). *n* será de la forma **u** *d*, donde **u** son todos los dígitos menos el último (es decir, **u** = *n*/10), y *d* es el último dígito (es decir *d* = n%10). Supongamos, asimismo, que hemos resuelto el problema para **u** = *n*/10 (llamada recursiva). Sea *c'* la solución a dicho sub-problema (es decir, la cantidad de números singulares menores que n/10):


Tenemos que ver cómo obtener la cantidad *c* de números singulares menores que *n* a partir de *c'* y de *n*:



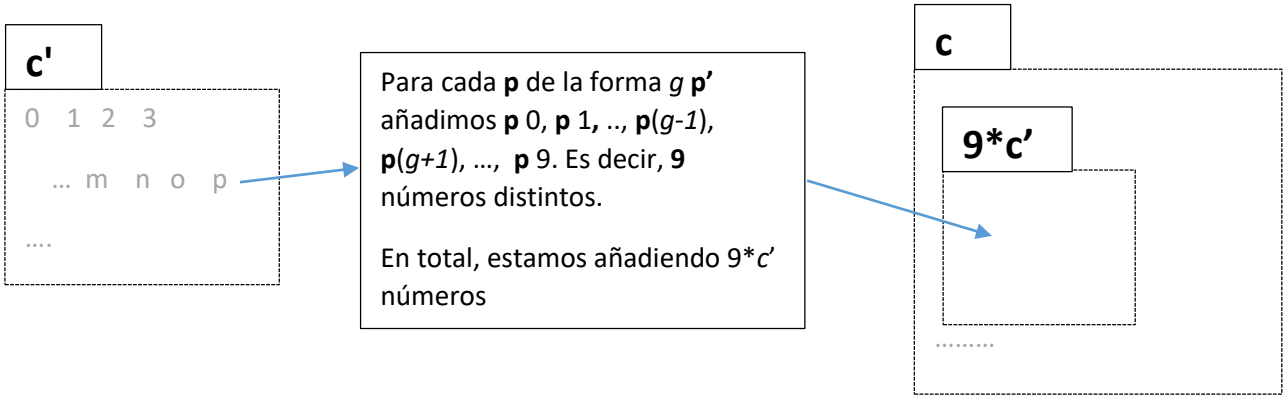

(La interrogación en este esquema representa la lógica de combinación del caso recursivo, que es la lógica que tenemos que determinar).

La mayoría de los números en la caja asociada a *c* puede obtenerse como sigue:

- Cogemos un número **p** de la caja asociada a **c'**
- Le ponemos un dígito *f* a final, que no coincida con el más significativo de **p**. Es decir, formamos el número **p** *f*
- Echamos ese número a la caja asociada a *c*.

¿Cuántos números podemos *echar*, con este método, a la caja asociada a *c*?:

- Para que los números de la forma **p** *f* que estamos echando a la caja sean singulares *f* no puede coincidir con el dígito más significativo de **p**.
- Es decir, si **p** es de la forma *g* **p'**, *f* podrá ser cualquier dígito, menos *g*. Tenemos, por tanto, 9 posibilidades para *f* (por ejemplo, si **p** = 687**,** *f* podrá ser el 0, el 1, el 2, el 3, el 4, el 5, el 7, el 8 y el 9: es decir, 9 posibles dígitos).
- Como esto lo podemos hacer para todos los números en la caja asociada a *c'*, podemos *echar* 9**c'* números a la caja asociada a *c*.



Observad que todos los números que están en la caja asociada a **9*c'** son números singulares menores que *n* de la forma **p** *f*, donde **p** es un número singular menor que *n*/10. O, dicho de otra forma, números singulares menores que *n* que **no** son de la forma (*n*/10) *f'*.

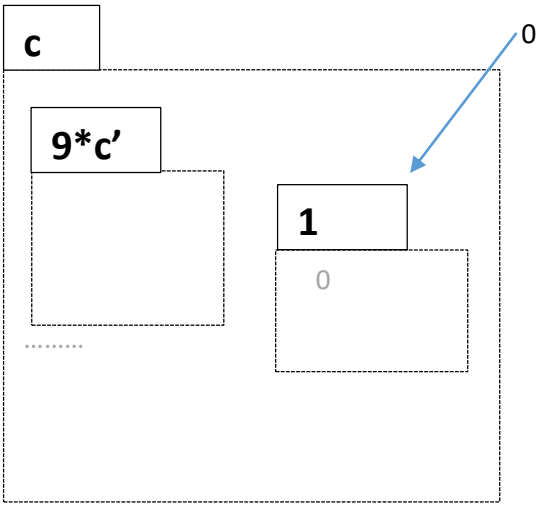
Pero, ¿la caja para **9*c'** contiene todos los números singulares menores que *n*/10 que **no** son de la forma (*n*/10) *f'*, o falta alguno?

**¡Falta el 0!**.

Efectivamente, con el procedimiento anterior, si partimos del 0, echaremos en la caja del **9*c'** el 1, el 2, el 3, el 4, el 5, el 6, el 7, el 8, y el 9. Y, si partimos de un número positivo, nunca podremos echar el 0 a la caja aplicando dicho procedimiento.

Por tanto, tendremos que echar explícitamente el 0 a la caja para *c*

.




(Observad que la cantidad asociada a la nueva caja que hemos añadido conteniendo únicamente el 0 es 1, ya que dicha caja contiene exactamente un número: el 0)

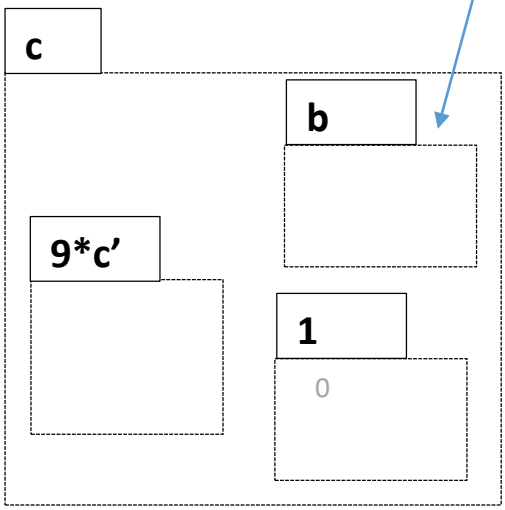
Con esto, hemos conseguido meter en la caja asociada a *c* todos los números singulares menores que *n* que **no** son de la forma (*n*/10) *f'*. ¿Qué nos falta para *completar* la caja?: meter todos los números singulares menores que *n* **que son** de la forma (*n*/10) *f'*. Para ello:

- Si (n/10) **no es singular**, tampoco lo será ningún número de la forma (*n*/10) *f'*. Por tanto, en este caso, no hay nada que añadir.
- Si (n/10) **es singular**, y recordando que *d* era el dígito menos significativo de *n*, es decir *d* = n%10, podremos añadir el (n/10)0, el (n/10)1, …., hasta el (n/10)(*d*-1), con la excepción del (n/10)*g* (siendo *g* el digito más significativo de n/10). Tenemos, por tanto, dos casos distintos:
    - n%10 es **menor o igual que el dígito más significativo de n/10**. En este caso, podemos añadir, exactamente, **n%10** nuevos números. Por ejemplo, si *n*=5674, los números a los que me refiero son: 5670, 5671, 5672, y 5673 (es decir, hay 4 = (5674%10) de estos números).
    - n%10 **es mayor que el dígito más significativo de n/10**. En este caso, habrá que quitar 1 a n%10, el correspondiente a poner al final dicho dígito más significativo. Podremos añadir, por tanto, **n%10-1** números. Por ejemplo, si

n=5678, los números a los que me refiero son: 5670, 5671, 5672, 5673, 5674, 5676, 5677 (es decir, hay 7 = (5678%10)-1 de estos números).

Sea *b* =

$$\begin{cases} n\%10 \text{ si } \frac{n}{10} \text{ es singular y n\%10 es menor o igual que el dígito más significativo de } \frac{n}{10} \\ n\%10 - 1 \; si \; \frac{n}{10} \; es \; singular \; y \; n\%10 \; es \; mayor \; que \; el \; dígito \; mas \; significativo \; de \; \frac{n}{10} \\ 0 \; en \; otro \; caso \end{cases}$$




**¡Y, con esto, hemos completado ya la caja para *c*!**.

Por tanto:

$$c = 9 * c' + 1 + \begin{cases} n\%10 \text{ si } \frac{n}{10} \text{ es singular y n\%10 es menor o igual que el dígito más significativo de } \frac{n}{10} \\ n\%10 - 1 \; si \; \frac{n}{10} \; es \; singular \; y \; n\%10 \; es \; mayor \; que \; el \; dígito \; mas \; significativo \; de \; \frac{n}{10} \\ 0 \; en \; otro \; caso \end{cases}$$

**¡Este es el resultado que debe devolver nuestro algoritmo, en caso de que n>9!**.

Lo difícil ya está hecho. Lo que nos queda ahora es añadir nuevos parámetros para poder calcular dicho valor (esto nos determinará cómo tiene que ser la **generalización** que vamos a utilizar). Para ello, necesitamos saber:

- **Si n/10 es o no singular**. Por tanto, aparte de devolver *c*, nuestro algoritmo deberá devolver si el número *n* recibido como parámetro es o no singular. Para ello,

**introduciremos en la generalización, un parámetro de salida adicional que indicará si el número *n* recibido como entrada es singular o no**.

- Por otra parte, necesitamos conocer también cuál es **el dígito más significativo de n/10**. Por tanto, **introduciremos, en la generalización, un segundo parámetro adicional de salida que indicará cuál es el dígito más significativo del número *n* recibido como entrada.**
- Con todo ello, tras resolver el problema para n/10, aparte de la cantidad *c*', tendremos también un booleano que nos determinará si n/10 es o no singular, así como, directamente, el dígito más significativo de n/10.

Con ello, la generalización (a la que podemos llamar `cuenta_singulares_menoresque`) tendrá los siguientes parámetros formales:

- Un parámetro de entrada *n*, el número para el que deseamos determinar la cantidad de singulares menores que *n*.
- Un parámetro de salida *c*: la cantidad de singulares menores que *n*.
- Un parámetro de salida *es_singular*: será cierto si *n* es singular, falso en otro caso.
- Un parámetro de salida *d_mas_sig*, que contendrá el dígito más significativo de *n.*

Como ya hemos realizado el trabajo difícil, el diseño por casos para la generalización es directo:

- Caso base: n<=9. En este caso:
  - *c*=n
  - *es_singular*=true (ya que cualquier número de un dígito lo es)
  - *d_mas_sig*=*n* (ya que si *n* tiene un solo dígito, *n* en sí mismo es el dígito más significativo de *n*)
- Caso *recursivo: n>9*
  - Resolvemos el problema para n/10. Sean *c*', *es_singular*' y *d_mas_sig'* los resultados devueltos (*c*': cantidad de números singulares menores que n/10; *es_singular*', *true* si n/10 es singular, false en otro caso; *d_mas_sig*': dígito más significativo de n/10)
  - Por el análisis ya realizado anteriormente

$$c = 9 * c' + 1 + \begin{cases} n\%10 \text{ si } es\_singular' \text{ y } n\%10 \leq d\_mas\_sig' \\ n\%10 - 1 \; si \; es\_singular' \; y \; n\%10 > \; d\_mas\_sig' \\ 0 \; en \; otro \; caso \end{cases}$$

  - *n* será singular si (i) n/10 es singular; y (ii) n%10 es distinto del dígito más significativo de n/10. Es decir, es_singular = es_singular' && (n%10 != d_mas_sig')
  - Por último, el dígito más significativo de *n* coincide con el más significativo de n/10: *d_mas_sig* = *d_mas_sig*'

Obsérvese que, en el caso recursivo, pueden utilizarse directamente los parámetros de salida *c*, *es_singular*, y *d_mas_sig* para dejar los resultados cuando se realiza la llamada recursiva. Si lo hacemos así, el análisis para el caso recursivo se modifica como sigue:

  - Resolvemos el problema para n/10. Dejamos los resultados en *c*, *es_singular* y *d_mas_sig*
  - *Multiplicamos c por 9, le sumamos 1, y le sumamos*

$$\begin{cases} n\%10 \text{ si } es\_singular \text{ y } n\%10 \leq d\_mas\_sig \\ n\%10 - 1 \; si \; es\_singular \; y \; n\%10 > \; d\_mas\_sig \\ 0 \; en \; otro \; caso \end{cases}$$

- es_singular = es_singular && (n%10 != d_mas_sig)
- Como el dígito más significativo de *n* coincide con el más significativo de n/10, no hay que hacer nada con *d_mas_sig* (ya tiene el dígito más significativo de *n*).

Por último, ¿cómo podemos realizar la inmersión del algoritmo original en términos de la generalización que hemos definido?:

- Basta invocar a la generalización con *n*, dejando los resultados en las variables locales *c*, *es_singular* y *d_mas_sig*
- … y devolver *c* como resultado

El código:

```
typedef long long tnum;
void cuenta_singulares_menoresque(tnum n, tnum& c, bool& es_singular,
                                  int& d_mas_sig) {
      if (n<=9) {
          c=n;
          es_singular=true;
          d_mas_sig=(int)n;
      }
     else {
        cuenta_singulares_menoresque(n/10, c, es_singular,d_mas_sig);
        c*=9;
        c++;
        if (es_singular) {
            if (n%10 <= d_mas_sig) {
                c += n%10;
            }
           else {
              c += (n%10 -1);
           }
        }
        es_singular = es_singular && (n%10 != d_mas_sig);
     }
}

tnum num_singulares_menoresque(tnum n) {
   tnum c;
   bool es_singular;
   int d_mas_sig;
   cuenta_singulares_menoresque(n, c, es_singular, d_mas_sig);
   return c;
}
```

(la razón para usar **long long** para representar tanto el número *n* como la cuenta *c* es, simplemente, poder manejar números más largos).