# Particiones de un número natural

Un conjunto de números naturales >0 es una partición de un número dado n, si la suma de dichos números es n. Es decir:

$$\{a_1, a_2, \dots, a_k\} \in Particiones(n) \Leftrightarrow \forall_i\, a_i > 0 \ \wedge \sum_i a_i = n$$

Lo que se desea es diseñar e implementar una función tal que, dado un número n, calcule el número de particiones distintas de dicho número. Es decir,

```
1 public int numPartitions(int nat) {
2   ¿?
3 }
```

El problema será encontrar una solución recursiva del mismo.

## Intentado buscar una recursión

Una estrategia que a veces funciona es la de intentar calcular manualmente los valores que ha de buscar la función y ver si podemos basar nuestro diseño en ese método de cálculo.
Otra estrategia que podemos aplicar cuando nos pidan calcular el número de veces que algo se puede hacer, es generar ese algo a contar de forma organizada.
Intentemos buscar las particiones de varios números, de la manera más organizada posible, ya que querremos poder luego programarla (y teniendo en cuenta que querremos usar una estrategia recursiva, es decir, una en la que el número der particiones de un número se calcule en base al número de particiones de otros números).
Empecemos[1]:

- f(1) = #{{1}}=1
- f(2) = #{{2}, {1+1}} = 2

¿Podemos intentar encontrar aquí una posible regla? La respuesta es que sí, las formas de sumar 2 se pueden dividir en dos categorías:

- Una en la que solamente usamos un número, es decir, {2}
- Otra en la que usamos varios números, en este caso, {1,1}

Fijaos en que en este último caso, los números que aparecen, son menores que 2. ¿Podemos intentar encontrar una regla para generarlos?

[1] Para simplificar la notación usaremos f(n) como el número de particiones del número n y #{c} como la cardinalidad del conjunto c.

### Primera aproximación

Una posible idea consiste en pensar en que si tenemos que n = p+q, cualquier combinación que sume p, sumada a una combinación que sume q, es una combinación que suma n.
Por ejemplo, como 10 = 4 + 6, una combinación que sume 4, por ejemplo {2,2} junto con una combinación que sume 6, por ejemplo {4,1,1}, forman una combinación {2,2,4,1,1} que suma 10.
Así que, podemos conjeturar que la recursión será:

$$f(n) = 1 + \sum_{k=1}^{n-1} f(k) * f(n-k) \quad , n > 1$$

¿Probamos?

- f(2) = 1 + #(1)*#(1) = 1+1*1 = 2
- f(3) = 1 + #(1)*#(2) + #(2)*#(1) = 1 + 1*2 + 2*1 = 5

Pero 3 se descompone como {3}, {2,1}, {1,1,1}, es decir, de 3 formas diferentes.

### Segunda aproximación

Para evitar repeticiones, si ya hemos considerado la descomposición de 3 como 2+1 ya no consideraremos la 1+2, es decir:

$$f(n) = 1 + \sum_{k=1}^{n\ div\ 2} f(k) * f(n-k) \quad , n > 1$$

¿Probamos?

- f(2) = 1 + f(1)*f(1) = 1 + 1*1 = 2
- f(3) = 1 + f(1)*f(2) = 1 + 1*2 = 3
- f(4) = 1 + f(1)*f(3) + f(2)*f(2) = 1 + 1*3 + 2*2 = 8

Pero 4 se descompone en, {4}, {3,1}, {2,2}, {2,1,1}, {1,1,1,1}. El problema es que la {1,1,1,1} y {2,2} se cuentan dos veces.
En resumen, nuestro problema consiste en encontrar formas de contar que eviten considerar varias veces la misma descomposición (en otras palabras, descomposiciones del problema que, por construcción, sean independientes entre sí). Cuando esto sucede, una estrategia común es añadir algún parámetro que permita distinguir unos subproblemas de otros.

### Posibilidad 1: distinguiendo particiones por tamaño

Una forma de sistematizar el conteo de particiones es por su tamaño (ya que una misma partición no puede tener dos tamaños diferentes). Si llamamos f(n,k) al número de particiones de n de tamaño k, tenemos que:

$$f(n) = \sum_{k=1}^{n} f(n,k)$$

Por lo que ahora el problema consistirá en buscar una recurrencia que nos permita calcular f(n,k) que es el número de particiones del número n usando k números positivos.
Como siempre debemos buscar casos simples y casos recursivos.

- f(n,k)=0 cuando k>n o k<=0
- f(n,k)=1 cuando k=n

Por lo que nos quedan los casos en los que k<n.

### ❖ *Propiedad 1: Particiones de menor tamaño*

¿Cómo se obtienen particiones de tamaño k a partir de particiones de tamaño k-1? Sumando un número.
Supongamos que tengo una partición de tamaño k-1 del número n, si le sumo el número $d_k$ obtengo una partición de tamaño k del número $n+d_k$.
Es decir,

$$n = \sum\{d_1, \dots, d_{k-1}\} \Leftrightarrow n + d_k = \sum\{d_1, \dots, d_k\}$$

o también

$$n = \sum\{d_1, \dots, d_k\} \Leftrightarrow n - d_k = \sum\{d_1, \dots, d_{k-1}\}$$

### ❖ *Propiedad 2: Particiones con sumandos menores*

Otra manera de manipular las particiones es cambiar alguno de los sumandos que aparecen en la descomposición. Es decir:

$$n = \sum\{d_1, \dots, (d_k + d)\} \Leftrightarrow n - d = \sum\{d_1, \dots, d_k\}$$

o también

$$n = \sum\{d_1, \dots, d_k\} \Leftrightarrow n - d = \sum\{d_1, \dots, (d_k - d)\}$$

siempre y cuando $d_k > d$.

Si todos los sumandos son mayores que d podemos aplicar varias veces la propiedad anterior y obtenemos

$$n = \sum\{d_1, \dots, d_k\} \Leftrightarrow n - k * d = \sum\{(d_1 - d), \dots, (d_k - d)\}$$

### ❖ *Aplicación al caso recursivo*

¿Podemos aprovecharnos de estas propiedadrd para encontrar una recurrencia? (En este punto conviene recordar que queremos obtener subproblemas que no contengan particiones en común, ya que no

queremos volver a caer en el error de sumar una misma partición varias veces).

- ¿Cuántas descomposiciones contienen al menos un 1?
  Por la propiedad 1 tantas como formas de obtener n-1 usando k-1 números, es decir, f(n-1, k-1).
- ¿Y cuantas no lo contienen? Si no contienen un 1, quiere decir que todos los sumandos son >1, por lo que por la propiedad 1 podemos restar un 1 a cada uno de ellos. Eso quiere decir que de éstas hay tantas como f(n-k, k).
- ¿Hay alguna otra posibilidad? No. O bien una partición contiene al menos un 1, o no lo contiene. ¡¡Ya hemos acabado!!

### ❖ *Recapitulando*

Si agrupamos todo lo que hemos desarrollado tenemos que:

$$f(n,k) = \begin{cases} 0, & n < k \; o \; k \leq 0 \\ 1, & n = k \\ f(n-1,k-1) + f(n-k,k), & n > k \end{cases}$$

Y expresando todo en Java quedaría:

```
1 public int numPartitions(int sum) {
2   // Entrada: sum > 0
3   int count = 0;
4   for (int numParts = 1; numParts <= sum; numParts++) {
5     count += numPartitions(sum, numParts);
6   }
7   return count;
8 }
9
10 public int numPartitions(int sum, int numParts) {
11   // Entrada: sum > 0
12   if (numParts <= 0 || numParts > sum ) {
13     return 0;
14   } else if (numParts == sum) {
15     return 1;
16   } else {
17     return numPartitions(sum-1, numParts-1) +
18            numPartitions(sum-numParts, numParts);
19   }
20 }
```

Fijaos en que, en esta solución, el "tamaño" del problema puede definirse como `sum + numParts`, por lo que en ambas llamadas recursivas los tamaños de los subproblemas son menores.

### Posibilidad 2: limitando el mínimo sumando de una partición

Otra forma de sistematizar el conteo, y evitar repeticiones, es fijar el valor mínimo que puede tener un sumando en una partición.
Es decir, g(n, k) = número de formas de sumar n con sumandos que son >=k.
En este caso el problema original f(n) es exactamente g(n,1) ya que, por definición, todos los sumandos serán >=1.
Queda como ejercicio buscar la recursión en este caso.