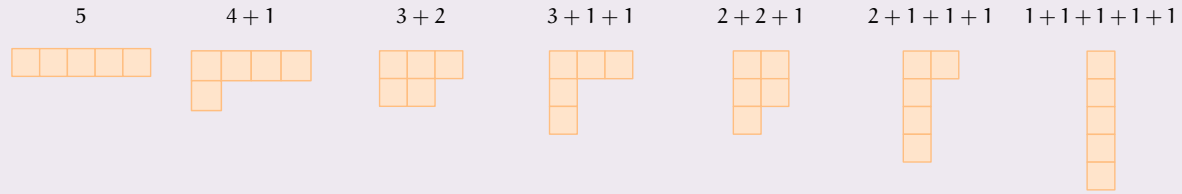
Particiones

La función $p(n)$ cuenta, para un número natural n, de cuántas maneras puede escribirse como suma de enteros positivos, sin tener en cuenta el orden de los sumandos.

Por ejemplo, hay $p(5) = 7$ particiones del número 5:



n	0	1	2	3	4	5	6	7	8	9	10	⋯
$p(n)$	1	1	2	3	5	7	11	15	22	30	42	⋯

http://oeis.org/A000041

El valor $p(0) = 1$ proviene de la toma en consideración de la partición vacía (la que no tiene ningún sumando) para el número $0 = \sum(\varnothing) = \sum_{i\in\varnothing} i$.

¿Cómo calculamos $p(n)$ a partir de n?

La estrategia que proponemos comienza haciendo más difícil el problema, generalizándolo a la evaluación de la función $p_k(n)$, definida como el número de particiones de n que tienen k partes (k sumandos) como mucho.

n \ k	0	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1	1
1		1	1	1	1	1	1	1	1
2		1	2	2	2	2	2	2	2
3		1	2	3	3	3	3	3	3
4		1	3	4	5	5	5	5	5
5		1	3	5	6	7	7	7	7
6		1	4	7	9	10	11	11	11
7		1	4	8	11	13	14	15	15
8		1	5	10	15	18	20	21	22

Cf. http://oeis.org/A026820

Por ejemplo, de los dibujos anteriores, solo hay $3 = p_2(5)$ (los tres primeros) que quepan en dos filas.

Esta tabla, destacada en un libro del siglo XVIII, se conoce como *tabla de Euler*.[1] La función $p_k(n)$ toma un valor positivo (es decir: hay alguna partición de n con k partes o menos) cuando $n \geq 0$ y $k > 0$, además de en el caso $n = k = 0$. En cualquier otra situación, definimos $p_k(n) = 0$.

1) *Contando con la capacidad para calcular $p_k(n)$ para parámetros cualesquiera, ¿de qué manera se puede obtener la cantidad de particiones de un número?*

La complicación del problema tiene como ventaja la posibilidad de recurrir a la siguiente propiedad:

[1] L. Euler: *Introductio in Analysin Infinitorum I* (1748), pp. 327-328.

n \ k	0	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1	1
1		1	1	1	1	1	1	1	1
2		1	2	2	2	2	2	2	2
3		1	2	3	3	3	3	3	3
4		1	3	4	5	5	5	5	5
5		1	3	5	6	7	7	7	7
6		1	4	7	9	10	11	11	11
7		1	4	8	11	13	14	15	15
8		1	5	10	15	18	20	21	22

$$p_k(n) = p_{k-1}(n) + p_k(n-k).$$

2) *Explica con tus palabras porque se cumple esta igualdad.*

Esto conduce a un algoritmo recursivo para la enumeración de las particiones:

cod_03_02.py

```
#! /usr/bin/env python3

def p(n):
    """Calcula el número de particiones
    de n.
    """

    return p_act(n, n)

def p_act(n, k):
    """Calcula el número de particiones
    de n con longitud acotada por k.
    """

    if n == k == 0:
        return 1
    elif n >= 0 and k >= 1:
        return p_act(n, k - 1)\
            + p_act(n - k, k)
    else:
        return 0

if __name__ == '__main__':
    import sys
    if len(sys.argv) > 1:
        n = int(sys.argv[1])
        print('p(%d) = %d' % (n, p(n)))
```

3) *Prueba este algoritmo con distintos valores del parámetro para hacerte una idea de su alcance.*

4) *Compáralo con el algoritmo que usa Sage:*

```
$ sage ↲

sage: Partitions(5).cardinality() ↲
7
sage: Partitions(1000).cardinality() ↲
24061467864032622473692149727991
sage: Partitions(10 ** 7).cardinality() ↲
920271755026...
```

Hay margen entonces para mejorar la solución de la izquierda.

Podríamos afinar las reglas de determinación de la función $p_k(n)$ para ahorrar algunos pasos de la recursión, pero esto tampoco nos conduce demasiado lejos.

Date cuenta de que

$$\forall k \geq n \ : \ p_k(n) = p_n(n).$$

```
def p_act(n, k):
    if n == k == 0:
        return 1
    elif n >= 0 and k >= 1:
        if n == 0 or k == 1:
            return 1
        elif k > n:
            return p_act(n, n)
        else:
            return p_act(n, k - 1)\
                + p_act(n - k, k)
    else:
        return 0
```

5) *Traduce los programas anterior a Scheme y da una cota sobre la complejidad espacial y temporal.*

6) *Siguiendo la estrategia de «memoization», convierte el algoritmo recursivo «ingenuo» en dos algoritmos* (iterativo/«bottom-up» *y* «top-down») *para el cálculo de las particiones de un número y prográmalos en Scheme.*

7) *Responde a las siguientes preguntas en términos de la complejidad especial y temporal para los nuevos algoritmos.*

   Hasta alcanzar $p(n)$ *con la estrategia* bottom-up, *¿cuántos términos de la tabla de Euler se calculan? Para cada uno de estos cálculos, ¿a cuántas valores previamente almacenados se recurre? ¿Cuántas operaciones aritméticas se emplean en la obtención de* $p(n)$*?*

Observa que no nos preguntamos por la cantidad de «operaciones máquina». Para completar el estudio de la complejidad del algoritmo, sería necesario acotar la talla de los resultados intermedios. Del resultado indicado al pie se deduce que esta talla se mantiene acotada por un polinomio.

8) *¿Permite algún ahorro en este caso la estrategia* top-down*?*

Para comprender el algoritmo tan eficiente que usa Sage,[2] hace falta sumergirse más en las matemáticas. En concreto, a partir de la identidad asintótica

Hardy y Ramanujan (1918)

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp(\pi\sqrt{2n/3}),$$

H. Rademacher obtuvo en 1938 una serie que aproxima la función de particiones y dedujo una cota en $O(\sqrt{n})$ para la cantidad de términos que garantiza la obtención del valor exacto.

[2] F. Johansson: «Efficient implementation of the Hardy-Ramanujan-Rademacher formula», LMS J. Comput. Math. **15**, 2012.