Detalle del Cómputo de la Representación de un Grafo de Partición

Samuel Alonso, 2019

Preliminares

Intentaré explicar el procedimiento sin entrar en los detalles matemáticos. El elemento fundamental del cómputo es un tipo de arreglo de la siguiente forma:

```
[k,[a_1, a_2, ... , a_n]]
```

donde k y los a_i son números naturales. La representación que se desea hallar es una arreglo de dichos arreglos:

```
rep = [
    [k_1,[a_11, a_12, ... , a_1n]],
    [k_2,[a_21, a_22, ... , a_2n]],
    ...
    [k_m,[a_m1, a_m2, ... , a_mn]]
]
```

A cada número natural n le corresponde un grafo de partición g_n, y por tanto una representación $r(g_n)$. Por el momento, abreviaremos $r(g_b)$ como r_n. Las representaciones satisfacen la relación de recurrencia dada por

$$r_n = n \oplus \sum_{k=1}^{\lambda_n} r_k * r_{n-k} \tag{2}$$

con

$$\lambda_n = \begin{cases} n/2, & n \text{ par} \\ (n-1)/2, & n \text{ impar} \end{cases} \tag{1}$$

y

```
r_1 = [
    [1,[1]]
]

r_2 = [
    [1,[2]],
    [1,[1,1]]
]
```

donde n simboliza el arreglo `[1,[n]]`, $\oplus$ equivale a la concatenación de representaciones (en pseudocódigo)

```
rep1 = [
    [k_1,[a_11, a_12, ... , a_1n]],
```

```
    [k_2,[a_21, a_22, ... , a_2n]],
    ...
    [k_m,[a_m1, a_m2, ... , a_mn]]
]

rep2 = [
    [q_1,[b_11, b_12, ... , b_1l]],
    [q_2,[b_21, a_22, ... , b_2l]],
    ...
    [q_s,[b_m1, a_m2, ... , b_sl]]
]

rep1 + rep2 = [
    [k_1,[a_11, a_12, ... , a_1n]],
    [k_2,[a_21, a_22, ... , a_2n]],
    ...
    [k_m,[a_m1, a_m2, ... , a_mn]],
    [q_1,[b_11, b_12, ... , b_1l]],
    [q_2,[b_21, a_22, ... , b_2l]],
    ...
    [q_s,[b_s1, a_s2, ... , b_sl]]
]
```

y $*$ equivale al producto (distributivo) entre representaciones, elemento a elemento, dado por la regla

```
elem1 = [k,[a_1, a_2, ... , a_n]]
elem2 = [q,[b_1, b_2, ... , b_m]]

elem1 * elem2 = [k*q, [a_1, a_2, ... , a_n, b_1, b_2, ... , b_m]]
```

asumiendo que `*` dentro del arreglo equivale al producto usual $(\cdot)$. Por ejemplo, la representación de 6 viene dada (en pseudocódigo) por

```
r(6) = [1,[6]] + r(1)*r(5) + r(2)*r(4) + r(3)*r(3)
```

Implementación en Python 3

Para la operación de suma $\oplus$ podemos usar la suma convencional de Python para arreglos:

```
def elem_sum(A,B): #A y B arreglos de la forma [ [k_1,[a_11, a_12, ... , a_1n]], ... , [k_m,[a_m1, a_m2, ... , a_mn]] ]
    return A+B
```

Para el producto $*$, definimos

```
def elem_product(A_elem,B_elem): #A_elem y B_elem elementos de la forma [k_1,[a_11, a_12, ... , a_1n]]
    C_scalar = A_elem[0]*B_elem[0]
    C_elem = [C_scalar,A_elem[1]+B_elem[1]]
    return C_elem
```

Entonces, la función que calcula la representación del grafo de partición de n puede ser escrita como

```
def r(n):
    if n == 1:
        return [[1,[1]]]
    if n == 2:
        return [[1,[2]],[1,[1,1]]]
    if n == 3:
        return [[1,[3]],[1,[1,2]],[1,[1,1,1]]]
    else:
        N = [[1,[n]]]
        for j in range(lam(n)):
            N = N + rep_product(r(j+1),r(n-(j+1)))
        return rep_simplify(N)
```

donde

```
def rep_product(A,B):
    len_a = len(A)
    len_b = len(B)
    C = len_a*len_b*[0]
    for i in range(len_a):
        for j in range(len_b):
            C[j*len_a + i] = (elem_product(A[i],B[j]))

    return rep_simplify(C)
```

y `rep_simplify` es una función que simplifica el producto distributivo, agrupando sobre los `[a_1, ... , a_n]` iguales.