Problema Knapsak

Problema de la mochila

También conocido por *Knapsack* por su nombre en inglés consiste en que, dados unos recursos limitados R (entero no negativo) y un conjunto de N elementos cada uno con una ganancia G_i (no negativo) y un costo L_i (entero no negativo), ¿Cuál es el subconjunto de elementos que maximiza las ganancias sin superar el limitante de recursos?

Fuente: https://openclipart.org/detail/313728/thief

Problema de la mochila

Ejemplo: para $R = 6$

i	1	2	3	4
G_i	12	18	30	44
L_i	1	2	3	4

Solución = {2,4}, con una ganancia de 62

¿Cuántas posibles soluciones tiene este problema? 2^N

¿Qué significa esto? Pues que por ejemplo para 10 elementos hay 1024 soluciones diferentes, para 20 hay 1048576, para 30 ≈ mil millones, etc.

```
max = 0
for i = 1 to N:
     c_i = 0
do:
     if true:
          process(c)
while nextCombination(c)

process(c):
     sum = 0, cost = 0
     for i = 1 to N:
          sum += c_i * P_i
          cost += c_i * L_i
     if sum > max and cost ≤ R:
          max = sum
          best = c
```

¿Cuál es la eficiencia de este algoritmo? $N2^N$

Combinaciones de un arreglo

En el anterior, así como en muchos otros problemas, se requiere encontrar todas las combinaciones de un determinado arreglo de base *b* (b=2: binario, b=3: trinario, etc.). Una alternativa en Java para ello es la siguiente:

```
void combine(int b, int n, String res, int s) {
        if (s == n) {
            //Procesar la combinación almacenada en el String res
            //Ejemplo: System.out.println(res);
        } else {
            for (int i = 0; i < b; i++) {
                combine(b, n, res + i + " ", s + 1);
            }
        }
}
```

Y en Python sería:

```
from itertools import product
array = [0, 1] # 0, 1, ... b-1
p = product(array, repeat=4) # repeat = N
for c in p:
   process(c) #Por ejemplo print(c)
```

```
(0, 0, 0, 0)
(0, 0, 0, 1)
(0, 0, 1, 0)
...
(1, 1, 1, 1)
```