Dream team

Capacidad para practicar una apendicectomía, virtuosismo con la guitarra, celeridad en el desempeño de la carpintería... y así hasta ciento treinta siete destrezas que ha detallado Shackleton en una lista y que aspira a cubrir con los integrantes de la expedición en ciernes.

Ya sea para para solventar las eventualidades que puedan acontecer en los hielos australes, ya para sostener los ánimos del grupo, el explorador pretende que entre todos los miembros de la partida que forme reúnan todas las habilidades de su lista (numerada del `0` al `136`),

Afortunadamente, los aspirantes a la aventura (treinta y dos candidatos que han contestado a los anuncios difundidos por la prensa londinense) despliegan un florido repertorio de talentos.

04_dat_01.txt

```
 0: 132  51 102 113
 1:  30  63  92  14 128  11   6 136  91  96  25  83
 2: 105 124  60  63  64 101 108  85 132  75
 3:   7  97 117 122  46 134   1  62  73  52 118  12 123  65 116  78
 4:  19  32   0  68  25  77 110  48 100 113
 5:  61  94 103   2 119  17
 6:  34  66  45  79  58  33  15  36  83  84 125  17
 7:  76  64 131  35  10  40  44  53 121  70  77  84
 8:  72 101  88  38 120  67  50  21 126  74 110 125
 9:  49  99  87  71  14  35  38  94  47  22  24  41
10:  16 129  82   3
11:  46 134   1  62  91 121 126 119 133   3  43 107
12:  56  31  28
13:  41 130
14: 108 114  98 106 104  48
15: 111  43 135
16:  73  52 110  12  96  70  74 104  27  82  31   8
17:  81   5   4  26 105 124  60  55  69  57  29  19  32   0  68  75
18:  34 127  47  59  42 133
19:  55  69  57  29   6  44  50 106  18 102
20:  22  95 109  66
21:  49  99  87  71  90  20 115  37 112   9  39  93  58  33  15  36
22:  79  23
23:  30  76  72  61 127 129 135  13 123  65 116  78
24: 107  13
25:  86   8
26:  85  89  80  18  27 100
27:   7  97 117 122  11  40  67  98  80  16  56  86
28:  90  20 115  37 128  10 120 103  59 109 111  23
29:  81   5   4  26  92 131  88 114  89  51  28
30:  24  45  54
31: 112   9  39  93 136  53  21   2  42  95  54 130
```

Los datos de los candidatos vienen codificados en un fichero de texto. Cada uno está representado en una línea, donde se recogen su número de orden y las habilidades de las que está dotado.

Utilizaremos esta notación:

C:	cjto. de candidatos
H_i:	cjto. de aptitudes de i
n:	n.º total de aptitudes

Se trata entonces de seleccionar un equipo $E \subseteq C$ que satisfaga la condición

$$\bigcup_{i \in E} H_i = [n],$$

donde $[n] = \{0, 1, \ldots, n-1\}$.

1) *Comprueba, antes de nada, que entre todos los aspirantes sí se reúnen todas las capacidades previstas por Shackleton, es decir,*

$$\bigcup_{i \in C} H_i = [n].$$

Bastaría entonces con tomar $E = C$. Pero Shackleton no se quiere llevar a todos. Al contrario, su objetivo es formar una cuadrilla tan reducida como sea posible, sin dejar de cumplir la condición mencionada anteriormente. Se enfrenta, entonces, al problema conocido como *set cover*.

En un primer momento, consideramos la **estrategia voraz** que, partiendo de la colección vacía de conjuntos, añade iterativamente un conjunto a la colección, escogiéndolo de tal manera que la cantidad de aptitudes incorporadas (que no lo estuvieran ya) sea máxima.[1] En símbolos: si P es un equipo que no cubre todas las destrezas, se incorpora un aspirante x que cumpla

$$\left|H_x \setminus \bigcup_{j\in P} H_j\right| = \max_{i\in C\setminus P} \left|H_i \setminus \bigcup_{j\in P} H_j\right| \qquad \left(= \max_{i\in C} \left|H_i \setminus \bigcup_{j\in P} H_j\right| \right).$$

Este algoritmo no garantiza una respuesta óptima. Recogemos de la bibliografía de la asignatura un par de cotas para la razón entre el tamaño de la colección resultante y el de *la* solución más pequeña:

$$\sum_{i=1}^{H} 1/i \;<\; 1 + \ln(H), \quad \text{donde } H = \max_{i\in C} |H_i|$$

T. Cormen, Ch. Leiserson, R. Rivest y C. Stein: *Introduction to algorithms*, 3.ª ed., Theorem 35.4.

$$\ln(n), \quad \text{donde } n = \left|\bigcup_{i\in C} H_i\right|$$

S. Dasgupta, Ch. Papadimitriou y U. Vazirani: *Algorithms*, p. 146

04_cod_02.py

```
def voraz(fich):
    p = Parcial_vrz(fich)
    while not p.es_completa():
        p.amplía_voraz()
    return p

resp = voraz('04_dat_01.txt')
print(resp, resp.coste())
```

El código planteado[2] se apoya en la clase `Parcial_vrz`. Esta deriva de `Parcial`, cuyos objetos (*instances* con la terminología de Python) representan colecciones de conjuntos. La clase `Parcial` cuenta con las siguientes funciones miembro (*cf.* `dia_00_cod_20.py`):

`__init__(self, fich)`

Construye una colección vacía. Carga los datos de los conjuntos disponibles para añadir a la colección a partir de un fichero de entrada como `04_dat_01.txt`.

`__repr__(self)`

Representa una colección en forma de texto.

`es_completa(self)`

Determina si una colección reúne o no todas las habilidades. Al decir «todas», nos referimos a la unión de las habilidades de todos los candidatos, que tomamos como universo: no insistimos en la comprobación del apartado (1).

`coste(self)`

Devuelve la cantidad de conjuntos de que consta la colección.

Por su parte, los objetos de la clase `Parcial_vrz` están provistos del siguiente método:

[1] Observa que la expresión ***el*** *conjunto que más habilidades nuevas aporta* no es precisa, puesto que puede haber varios que tengan la misma cantidad de elementos ausentes de la colección. En función de estas elecciones, el proceso puede dar lugar a respuestas con tamaños distintos. Para que esta estrategia voraz defina un algoritmo determinista, es necesario concretar la manera de «desempatar».

[2] En Python 3, los identificadores pueden contener algunos caracteres externos al repertorio ASCII (PEP 3131, documentación del lenguaje). Entiéndase nuestra utilización de este recurso como una exposición de esta característica más que como un respaldo al empleo de identificadores exóticos, que es una manera fácil de granjearse una fuerte desaprobación.

`amplía_voraz(self)`

Añade a la colección un conjunto, produciendo la mayor incorporación de habilidades que sea posible.

2) *Escribe un módulo que defina la clase `Parcial` y otro para `Parcial_vrz`, de modo que el programa `04_cod_02.py` funcione correctamente.*
De seguir ese algoritmo voraz, ¿cuántos aspirantes reclutaría Shackleton para su partida? Deduce de esta respuesta una cota inferior para el tamaño de las soluciones.

3) *Complete el código del fichero `Voraz.rkt`, utilizando solamente listas. Explique, si la hubiera, cualquier perdida de rendimiento del algoritmo en comparación con la implementación en Python.*

El problema *set cover* es NP-completo.[3] Intentaremos resolver el caso propuesto mediante un algoritmo de ramificación y poda, siguiendo el mismo esquema que hemos utilizado para el problema del viajante (*cf.* `dia_00_cod_23.py`):

04_cod_08.py

```
infinito = float('inf')

def lanza(p, mj=None, t_mj=infinito):
    ct = p.cota()
    fondo = p.es_completa()
    escribe_línea(p, ct, t_mj, fondo, mejora)
    if ct < t_mj:
        if fondo:
            mj = p
            t_mj = ct
        else:
            for ap in p.amplía():
                mj, t_mj = lanza(ap, mj, t_mj)
    return mj, t_mj

p0 = Parcial_ct('04_dat_01.txt')
resp = lanza(p0)
print(resp, resp.coste)
```

Para ello, empleamos la clase `Parcial_ct` (derivada de `Parcial`), cuyas «instancias»[4] cuentan con el método adicional `cota`, que calcula, para una colección, una cota inferior para el tamaño de las soluciones que la contienen. Cuando `p` es una colección solución (es decir, que presenta todas las habilidades), la salida de `p.cota()` debe coincidir con la de `p.coste()`.

El programa utiliza también la función `amplía`, que genera las posibles ampliaciones de una colección de conjuntos.

4) *Completa la implementación de las clases `Parcial` y `Parcial_ct`, de modo que el programa anterior funcione correctamente.*
Calcula un grupo con un número de integrantes mínimo para la expedición antártica.

5) *Estudia el problema con los datos de entrada dados en el fichero `04_dat_02.txt`, con sesenta y seis candidatos y doscientas ochenta y dos habilidades requeridas por Shackleton.*

Los dos ficheros de datos facilitados con esta hoja no están tomados al azar, como los que escribe el programa `04_cod_09.py`.

6) *Compara, utilizando datos de entrada generados aleatoriamente, los resultados del algoritmo de tipo* branch and bound *con los del algoritmo voraz que has completado en (2).*

7) *Como puntos extra, el alumno podrá implementar el mismo código en scheme, completando el fichero `RyP.rkt`. El algoritmo deberá devolver la solución en un tiempo razonable.*

[3] Más precisamente, su versión decisional lo es. Del problema de optimización puede decirse, eso sí, que es NP-duro.
[4] Valga el barbarismo.