1. Una academia de inglés tiene n alumnos a ser repartidos en m grupos (n múltiplo de m). Cada grupo tiene distinto horario y profesor. De cada alumno se conoce la afinidad que tiene para pertenecer a cada uno de los grupos (valor entero en el rango [0,5]). Se desea conocer el reparto de alumnos en grupos, de forma que todos los grupos deben tener el mismo número de alumnos, maximizando la afinidad total conseguida para todos los alumnos, y teniendo en cuenta que no está permitido asignar un alumno a un grupo para el que presente afinidad 0.

$$max\ \Sigma_{i=0,j=0}^{n-1,m-1}\ a(i,j)x_{ij}$$

$$\Sigma_{i=0|a(i,j)>0}^{n-1} x_{ij} = \frac{n}{m},\ \ j \in [0,m)$$

$$\Sigma_{j=0}^{m-1} x_{ij} = 1,\ \ i \in [0,n)$$

$$bin\ \ x_{ij},\quad\ \ i \in [0,n), j \in [0,m)$$

```
head section


Integer getNGrupos()
Integer getNAlumnos()
Integer getNAfinidades()
Integer afinidadPorIndice(Integer i, Integer j)

Integer nGrupos = getNGrupos()
Integer alumnos = getNAlumnos()
Integer afinidades = getNAfinidades()

goal section

max sum(afinidadPorIndice(i, j) x[i, j], i in 0 .. alumnos, j in 0 .. afinidades)

constraints section

sum(x[i, j], j in 0 .. afinidades) = 1, i in 0 .. alumnos
sum(x[i, j], i in 0 .. alumnos | afinidadPorIndice(i, j) > 0) = nGrupos, j in 0 .. afinidades

bin

x[i, j], i in 0 .. alumnos, j in 0 .. afinidades
```

2. Un bufete de abogados cuenta con un equipo de n personas que deben analizar m casos relacionados entre sí (m>=n), y deben terminar dicho análisis global lo antes posible para lo que trabajarán en paralelo. Cada caso será analizado por un único abogado, y cada abogado puede analizar varios casos. Se conoce el tiempo (en horas) que se estima que tarda cada abogado en analizar cada caso concreto (dicho tiempo puede diferir para cada caso en función de qué abogado realice el análisis). Determine cuál es la mejor asignación de casos a abogados para conseguir el objetivo indicado (terminar de analizar todos los casos lo antes posible).

$$\min T$$

$$\sum_{i=0}^{n-1} x_{ij} = 1, \quad j \in [0, m-1]$$

$$\sum_{j=0}^{m-1} x_{ij}\, c(i,j) \leq T, \quad i \in [0, n-1]$$

$$bin\, x_{ij}, \quad i \in [0, n-1], \quad j \in [0, m-1]$$

```
head section


Integer getNHoras()
Integer getNAbogados()
Integer tiempoPorIndice(Integer i, Integer j)

Integer casos = getNHoras()
Integer abogados = getNAbogados()

goal section

min t[0]

constraints section

sum(x[i, j], i in 0 .. abogados) = 1, j in 0 .. casos
sum(tiempoPorIndice(i, j) x[i, j], j in 0 .. casos) - t[0] <= 0, i in 0 .. abogados

bin

x[i, j], i in 0 .. abogados, j in 0 .. casos
```

3. Se tienen n productos, cada uno de los cuales tiene un precio y presenta una serie de funcionalidades (el mismo producto puede tener más de una funcionalidad). Se desea diseñar un lote con una selección de dichos productos que cubran un conjunto de funcionalidades deseadas entre todos productos seleccionados al menor precio.

$$\min \sum_{i=0}^{n-1} p_i x_i$$

$$(\sum_{i \in \varphi(j)} x_i) \geq 1, \quad j \in [0, m)$$

$$bin\, x_i, \quad i \in [0, n)$$

```
head section


Integer getNProductos()
Double getPrecio(Integer i)
Integer getNFuncionalidades()
Boolean contiene(Integer producto, Integer funcionalidad)

Integer productos = getNProductos()
Integer funcionalidades = getNFuncionalidades()

goal section

min sum(getPrecio(i) x[i], i in 0 .. productos)

constraints section

sum(x[i], i in 0 .. productos | contiene(i, j)) >= 1, j in 0 .. funcionalidades

bin

x[i], i in 0 .. productos
```

4. Dado un conjunto de enteros determinar si puede particionarse en tres subconjuntos de manera que la suma de elementos en los tres subconjuntos sea la misma, y que el tamaño de uno de ellos sea lo menor posible.

$$\min \sum_{i=0|j=0}^{n-1} x_{ij}$$

$$\sum_{j=0}^{2} x_{ij} = 1, \quad i \in [0, n-1]$$

$$\sum_{i=0}^{n-1} x_{ij} e_i = \frac{1}{3} \sum_{i=0}^{n-1} e_i, \quad j \in \{0,1,2\}$$

$$bin\ x_{ij}, \quad i \in [0, n-1], j \in \{0,1,2\}$$

```
head section


Integer getSumatorio()
Integer getSizeConjunto()
Integer elemento(Integer i)

Integer size = getSizeConjunto()

goal section

min sum(x[i, 0], i in 0 .. size)

constraints section

sum(x[i, j], j in 0 .. 3) = 1, i in 0 .. size
sum(elemento(i) x[i, j], i in 0 .. size) = getSumatorio(), j in 0 .. 3

bin

x[i, j], i in 0 .. size, j in 0 .. 3
```