## Algoritmos de optimización - Seminario

Nombre: Marco Antonio Rumiche Pingo

URL: https://colab.research.google.com/drive/1ff9mxg9s5LTZMW8Kvm4VCCz61VGG3Lbf

https://github.com/rumichem/AlgoritmosdeOptimizacion-Seminario/tree/main/MARCO\_ANTONIO\_RUMICHE\_PINGO\_Seminario\_Algoritmos.ipynb

#### Problema:

- > 1. Sesiones de doblaje
- >2. Organizar los horarios de partidos de La Liga
- >3. Combinar cifras y operaciones

Descripción del problema:(copiar enunciado)

#### Problema 1. Organizar sesiones de doblaje(I)

• Se precisa coordinar el doblaje de una película. Los actores del doblaje deben coincidir en las tomas en las que sus personajes aparecen juntos en las diferentes tomas. Los actores de doblaje cobran todos la misma cantidad por cada día que deben desplazarse hasta el estudio de grabación independientemente del número de tomas que se graben. No es posible grabar más de 6 tomas por día. El objetivo es planificar las sesiones por día de manera que el gasto por los servicios de los actores de doblaje sea el menor posible. Los datos son:

Número de actores: 10 Número de tomas : 30

Actores/Tomas: https://bit.ly/36D8luK

- 1 indica que el actor participa en la toma
- 0 en caso contrario

![image.png](attachment:94e35157-52ee-4717-afb3-876a3f3faedc.png)

(\*) La respuesta es obligatoria

## 1.1 (\*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

#### Respuesta:

Si no se tienen en cuenta las restricciones (es decir, si ignoramos tanto el límite de 6 tomas por día como la compatibilidad de actores), entonces el problema se reduce a calcular todas las formas posibles de agrupar las 30 tomas en días sin restricciones.

En este caso, cada toma puede asignarse a cualquier día, y el número total de formas de distribuir n tomas en k días es equivalente al número de funciones de un conjunto de n elementos (tomas) a un conjunto de k elementos (días).

- 1. Si el número de días k es fijo:
- El número de formas es k<sup>30</sup> (cada toma tiene k opciones).
- Pero como queremos optimizar k, no es útil.
- 2. Si k es variable (desde 1 hasta 30 días):
- El número total de formas es el número de particiones de un conjunto de 30 elementos, conocido como el 30-ésimo número de Bell (B<sub>20</sub>).
- B<sub>30</sub> es un valor extremadamente grande:

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

- 3. Para obtener los primeros números de Bell son:  $B_0=1$ ,  $B_1=1$ ,  $B_2=2$ ,  $B_3=5$ ,  $B_4=15$ ,....
  - Para n=30, el valor exacto es:

 $B_{30} = 8462580824584107501473$ 

```
B_{30} \approx 8.46 \times 10^{23}
```

A continuación el algoritmo para realizar la comprobación del resultado:

```
In [13]: #
# Algoritmo para el calculo del números bell
# Referencia: https://es.wikipedia.org/wiki/N%C3%BAmero_de_Bell
#

from math import comb
from sympy import bell # usando funcion bell sólo para la comprobación del algoritmo

def bell_number(n):
    bell = [0] * (n+1)
    bell[0] = 1

for i in range(1, n+1):
    bell[i] = 0
    for j in range(i):
        bell[i] += comb(i-1, j) * bell[j]

    return bell[n]

print("Cálculo del número de Bell con el algoritmo, B30 = ", bell_number(30))
print("Cálculo del número de Bell con la función, B30 = ", bell[30))
```

### 1.2 ¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta:

Tenemos lo siguientes datos:

- Se tiene 30 tomas.
- Solo se pueden grabar máximo 6 tomas por día.
- No hay mínimo: los días pueden tener entre 1 y 6 tomas.
- El objetivo es minimizar el coste total, definido como el número total de actores únicos que se presentan cada día, sumado para todos los días.

Por lo tanto:

```
Total de formas de dividir 30 elementos en subconjuntos de tamaños: 1≤k≤6,para cada grupo i
```

```
donde: \sum k_i = 30
```

Para 30 tomas y máximo 6 tomas por día:

Queremos contar todas las secuencias de enteros positivos [a1, a2, .... ak], tales que:

Cálculo del número de Bell con el algoritmo, B30 = 846749014511809332450147 Cálculo del número de Bell con la función, B30 = 846749014511809332450147

```
a1 + a2 + ... + ak = 30
```

Cada composición representa una forma válida de distribuir tomas por días, como:

```
[6,6,6,6,6] \rightarrow 5 días, 6 tomas cada uno [5,5,5,5,5,5] \rightarrow 6 días de 5 tomas [6,6,6,6,5,1] \rightarrow 6 días de diferente tamaño
```

Cada una es una forma distinta de agrupar las tomas por día (sin saber aún qué tomas van en cuál grupo).

#### Por lo que existe 253 particiones posibles de composiciones sin orden.

Ahora las composiciones sin importar el orden, o más precisamente:

Todas las formas de particionar 30 como suma de enteros de 1 a 6

Pero donde el orden de los sumandos no importa

```
[6,6,6,6,6] y [6,6,6,6,6] \rightarrow una sola
```

[6,6,5,5,4,4], [5,6,4,6,4,5], etc. → una sola

- Este es el conteo de composiciones únicas
- Se parece más al concepto de particiones con sumandos acotados

Por lo tanto, las composiciones con orden: cada orden diferente cuenta es = 437,513,522

A continuación el algoritmo para realizar la comprobación del resultado:

```
In [15]: def contar_composiciones(n, k):
             Cuenta el número de formas en que se puede escribir el número `n`
             como suma de enteros positivos entre 1 y `k`, donde el orden importa.
             dp = [0] * (n + 1)
             dp[0] = 1 # Solo hay una forma de componer 0: usando ningún número
             for i in range(1, n + 1):
                 for j in range(1, k + 1):
                     if i - j >= 0:
                        dp[i] += dp[i - j]
             return dp[n]
         # Parámetros
         total\_tomas = 30
         max_tomas_por_dia = 6
         # Resultado
         total_composiciones = contar_composiciones(total_tomas, max_tomas_por_dia)
         print(f"Total de formas de agrupar {total_tomas} tomas con máximo {max_tomas_por_dia} por día:", total_composiciones)
```

Total de formas de agrupar 30 tomas con máximo 6 por día: 437513522

Modelo para el espacio de soluciones

# 1.3 (\*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al principio y veas la necesidad de cambiar, arguentalo)

Respuesta:

Se requiere identificar la estructura de datos óptima usando tuplas y backtracking para agrupar 30 tomas (máx. 6 por día) y minimizar los días de desplazamiento de actores.

Poder detectar si una solución parcial es prometedora reducimos el modelo del espacio de la siguiente manera:

Día | Tomas | Actores involucrados

Total de actores desplazados (suma por día) = 2 + 2 + 2 = 6

Dentro de la codificación el modelo sería algo asi:

Según el modelo para el espacio de soluciones

## 1.4 (\*)¿Cual es la función objetivo?

Respuesta:

Tenemos las siguientes variables:

- T={1,2,...,30}: conjunto de tomas
- A={1,2,...,10}: conjunto de actores
- D={1,2,...,d}: conjunto de días de grabación
- $td \in \{0,1\}$ :

xtd=1 si la toma t es asignada al día d; 0 en caso contrario

•  $yad \in \{0,1\}$ :

yad = 1 si el actor a participa en alguna toma del día d; 0 en caso contrario

•  $Pta \in \{0,1\}$ : matriz de participación Pta = 1 si el actor a participa en la toma t; 0 en caso contrario

#### **Función Objetivo:**

Minimizar el costo total de desplazamiento de actores a lo largo de los días:

![image.png](attachment:17e271ed-c60f-46db-9eeb-731edf77977a.png)

1. Cada toma debe ser asignada a un único día:

$$\sum_{d \in D} x_{td} = 1 \quad \forall t \in T$$

2. No puede haber más de 6 tomas por día:

$$\sum_{t \in T} x_{td} \leq 6 \quad \forall d \in D$$

3. Un actor solo se considera presente un día si participa en alguna toma asignada a ese día:

$$y_{ad} \geq x_{td} \cdot P_{ta} \quad \forall a \in A, \ \forall t \in T, \ \forall d \in D$$

(Esto garantiza que si el actor participa en una toma asignada al día  $d_i$  entonces yad = 1)

La función objetivo minimiza el número de días que cada actor debe desplazarse Las restricciones aseguran que:

- Cada toma se graba exactamente una vez
- Ningún día tiene más de 6 tomas
- Se activa el costo de un actor si aparece en cualquier toma de ese día

## 1.5 (\*); Es un problema de maximización o minimización?

Respuesta

Este es un problema de minimización. La suma total del número de veces que los actores deben desplazarse al estudio, es decir:

$$\min \sum_{d \in D} \sum_{a \in A} y_{ad}$$

Donde yad=1 si el actor a trabaja el día d y 0 en caso contrario.

Cada actor cobra por día que se presenta. Por tanto, el objetivo es:

- Asignar las tomas a los días (máx. 6 por día)
- De modo que los actores tengan que desplazarse el menor número de días posible para reducir el que se tenga que desplazar al set de rodaje, porque los actores cobran por día y el objetivo es obtener el mínimo coste del rodaje.

## 2.1 Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

```
import numpy as np
{\it from} itertools {\it import} combinations
# Matriz de 30 tomas x 10 actores
# -----
tomas = np.array([
   [1,1,1,1,1,0,0,0,0,0],
    [0,0,1,1,1,0,0,0,0,0],
    [0,1,0,0,1,0,1,0,0,0],
    [1,1,0,0,0,0,1,1,0,0],
    [0,1,0,1,0,0,0,1,0,0],
    [1,1,0,1,1,0,0,0,0,0],
    [1,1,0,1,1,0,0,0,0,0],
    [1,1,0,0,0,1,0,0,0,0],
    [1,1,0,1,0,0,0,0,0,0]
    [1,1,0,0,0,1,0,0,1,0],
    [1,1,1,0,1,0,0,1,0,0],
    [1,1,1,1,0,1,0,0,0,0]
    [1,0,0,1,1,0,0,0,0,0],
    [1,0,1,0,0,1,0,0,0,0],
    [1,1,0,0,0,0,1,0,0,0],
    [0,0,0,1,0,0,0,0,0,1],
    [1,0,1,0,0,0,0,0,0,0],
    [0,0,1,0,0,1,0,0,0,0],
    [1,0,1,0,0,0,0,0,0,0],
    [1,0,1,1,1,0,0,0,0,0],
    [0,0,0,0,0,1,0,1,0,0],
    [1,1,1,1,0,0,0,0,0,0],
    [1,0,1,0,0,0,0,0,0,0],
    [0,0,1,0,0,1,0,0,0,0],
    [1,1,0,1,0,0,0,0,0,1],
    [1,0,1,0,1,0,0,0,1,0],
    [0,0,0,1,1,0,0,0,0,0]
    [1,0,0,1,0,0,0,0,0,0],
    [1,0,0,0,1,1,0,0,0,0],
    [1,0,0,1,0,0,0,0,0,0],
])
num_tomas = tomas.shape[0]
tomas_asignadas = [False] * num_tomas
dias = []
# Funciones
def actores_en_grupo(grupo_indices):
   return np.any(tomas[list(grupo_indices)], axis=0)
def costo(grupo_indices):
   return int(np.sum(actores_en_grupo(grupo_indices)))
# Agrupación día por día
while not all(tomas_asignadas):
   disponibles = [i for i in range(num_tomas) if not tomas_asignadas[i]]
   tam_grupo = min(6, len(disponibles))
   mejor_grupo = None
   mejor_costo = float('inf')
    for grupo in combinations(disponibles, tam_grupo):
        c = costo(grupo)
        if c < mejor_costo:</pre>
           mejor_costo = c
           mejor_grupo = grupo
    # Marcar como asignadas
    for i in mejor_grupo:
       tomas_asignadas[i] = True
    dias.append(sorted(mejor_grupo))
# Resultados
coste_total = 0
print("\nPlanificación de grabación:")
print("----")
for i, grupo in enumerate(dias):
   actores = actores_en_grupo(grupo)
```

#### Planificación de grabación:

```
Día 1: Tomas [14, 17, 18, 19, 23, 24] -> Actores requeridos: 3
Día 2: Tomas [2, 13, 20, 27, 28, 30] -> Actores requeridos: 4
Día 3: Tomas [1, 3, 6, 7, 9, 15 ] -> Actores requeridos: 6
Día 4: Tomas [4, 5, 8, 12, 21, 22 ] -> Actores requeridos: 7
Día 5: Tomas [10, 11, 16, 25, 26, 29] -> Actores requeridos: 9

Resumen:
------
Total de días de grabación : 5
Total de desplazamientos de actores: 29
```

Validación completa: Todas las tomas se agruparon correctamente por días.

## 2.2 Calcula la complejidad del algoritmo por fuerza bruta

Respuesta:

El cálculo que se realiza es de Orden factorial, O(n!)

El algoritmo intenta explorar todas las formas posibles de agrupar n=30 tomas en días, donde cada día puede tener hasta 6 tomas.

Esto equivale a un problema de particionamiento de conjuntos con un límite superior por grupo (máx. 6 elementos por grupo), lo que se aproxima a una exploración de todas las permutaciones posibles de tomas asignadas a días.

#### Ejemplo:

Para solo 6 tomas, ya se evaluaron 2,350 combinaciones.

Para 30 tomas, hay más de 400 millones de formas posibles de agruparlas en días de hasta 6 tomas

El algoritmo en su forma actual tiene orden factorial, ya que explora (en el peor caso) todas las permutaciones y particiones posibles para encontrar el óptimo exacto.

## 2.3 (\*)Diseña un algoritmo que mejore la complejidad del algoritmo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta

Respuesta:

El objeto es agrupar las tomas del doblaje en días, con un máximo de 6 tomas por día, de forma que el número total de actores distintos por día sea mínimo. Cada actor cobra por día, por lo que se desea minimizar el total de desplazamientos de actores.

#### **Entradas:**

• matriz\_tomas: matriz binaria de tamaño T × A

(T = número de tomas, A = número de actores) Donde matriz\_tomas[i][j] = 1 si el actor j participa en la toma i.

• max\_tomas\_dia: cantidad máxima de tomas por día (6).

#### Salida:

- Agrupación de tomas por días (listas de índices de tomas).
- Total de desplazamientos de actores (suma de actores distintos por día).

#### **PASOS DEL ALGORITMO:**

```
Paso 1: Agrupación Greedy (voraz)
```

Objetivo: Agrupar tomas de forma que cada grupo de hasta 6 tomas tenga la menor cantidad de actores posible.

```
psudocódigo:
```

```
mientras existan tomas sin asignar:

para r = 1 hasta max_tomas_dia:

para cada combinación posible de r tomas no asignadas:

calcular actores distintos involucrados

si es mejor que el mejor grupo hasta ahora:

guardar grupo como el mejor

añadir el mejor grupo como un nuevo día

eliminar las tomas del grupo del conjunto de tomas disponibles
```

#### Paso 2: Postprocesamiento Local

Objetivo: Mejorar la solución moviendo tomas entre días si reduce el número total de actores.

#### Pseudocódigo:

```
mientras se pueda mejorar:

para cada par de días distintos (i, j):

para cada toma en el día i:

si el día j tiene espacio (menos de 6 tomas):

mover la toma del día i al día j temporalmente

calcular el nuevo número total de actores de ambos días

si mejora (disminuye), hacer el cambio permanente
```

#### Paso 3: Resultado

Imprimir:

- Para cada día: tomas incluidas y actores requeridos.
- Total de días de grabación.
- Total de desplazamientos de actores.

#### Complejidad Estimada

- Greedy (combinaciones hasta 6 tomas):  $O(n^6 \times a)$
- Postprocesamiento local: O(n<sup>2</sup> × a)
- Total: O(n<sup>6</sup>) (orden polinomial alto)

#### Ventajas frente a Fuerza Bruta:

- Evita calcular todas las particiones posibles (orden exponencial o factorial).
- Genera una solución razonablemente buena en tiempo aceptable.
- Fácil de escalar y adaptar a nuevas restricciones.

```
In [19]: from itertools import combinations
         from copy import deepcopy
         # Matriz de 30 tomas x 10 actores
         matriz tomas = [
             [1,1,1,1,1,0,0,0,0,0,0],
             [0,0,1,1,1,0,0,0,0,0],
             [0,1,0,0,1,0,1,0,0,0],
             [1,1,0,0,0,0,1,1,0,0],
             [0,1,0,1,0,0,0,1,0,0],
              [1,1,0,1,1,0,0,0,0,0],
             [1,1,0,1,1,0,0,0,0,0],
             [1,1,0,0,0,1,0,0,0,0],
             [1,1,0,1,0,0,0,0,0,0],
             [1,1,0,0,0,1,0,0,1,0],
             [1,1,1,0,1,0,0,1,0,0],
             [1,1,1,1,0,1,0,0,0,0],
             [1,0,0,1,1,0,0,0,0,0],
             [1,0,1,0,0,1,0,0,0,0],
             [1,1,0,0,0,0,1,0,0,0],
             [0,0,0,1,0,0,0,0,0,1],
              [1,0,1,0,0,0,0,0,0,0],
             [0,0,1,0,0,1,0,0,0,0],
```

```
[1,0,1,0,0,0,0,0,0,0],
    [1,0,1,1,1,0,0,0,0,0],
    [0,0,0,0,0,1,0,1,0,0],
    [1,1,1,1,0,0,0,0,0,0,0]
    [1,0,1,0,0,0,0,0,0,0],
    [0,0,1,0,0,1,0,0,0,0],
    [1,1,0,1,0,0,0,0,0,0,1],
    [1,0,1,0,1,0,0,0,1,0],
    [0,0,0,1,1,0,0,0,0,0],
    [1,0,0,1,0,0,0,0,0,0],
    [1,0,0,0,1,1,0,0,0,0],
    [1,0,0,1,0,0,0,0,0,0],
MAX TOMAS POR DIA = 6
# Calcula actores involucrados en un grupo de tomas
def actores_para(grupo):
   actores = set()
   for toma in grupo:
       for i, val in enumerate(matriz_tomas[toma]):
            if val == 1:
                actores.add(i)
    return actores
# Costo total = suma de actores únicos por día
def costo_total(agrupacion):
    return sum(len(actores_para(grupo)) for grupo in agrupacion)
# Paso 1: Algoritmo greedy básico
def greedy_agrupacion_tomas(matriz, max_tomas_dia):
    tomas_disponibles = set(range(len(matriz)))
   resultado = []
    while tomas_disponibles:
        mejor_grupo = None
       mejor_costo = float('inf')
        for r in range(1, min(max_tomas_dia, len(tomas_disponibles)) + 1):
            for grupo in combinations(tomas_disponibles, r):
                actores = actores_para(grupo)
                if len(actores) < mejor_costo:</pre>
                    mejor_grupo = grupo
                    mejor_costo = len(actores)
        resultado.append(list(mejor_grupo))
        tomas_disponibles -= set(mejor_grupo)
    return resultado
# Paso 2: Mejorar moviendo tomas entre días si reduce el costo total
def optimizar_postprocesado(dias, max_tomas_dia):
    mejorado = True
    while mejorado:
        mejorado = False
        for i in range(len(dias)):
            for j in range(len(dias)):
                if i == j or not dias[i]:
                for toma in dias[i]:
                    if len(dias[j]) < max_tomas_dia:</pre>
                        temp_i = dias[i].copy()
                        temp_j = dias[j].copy()
                        temp_i.remove(toma)
                        temp_j.append(toma)
                        nuevo_costo = len(actores_para(temp_i)) + len(actores_para(temp_j))
                        actual_costo = len(actores_para(dias[i])) + len(actores_para(dias[j]))
                        if nuevo_costo < actual_costo:</pre>
                            dias[i] = temp_i
                            dias[j] = temp_j
                            mejorado = True
                            break
                if mejorado:
                    break
            if mejorado:
                break
    return dias
# Ejecutar algoritmo
agrupacion_inicial = greedy_agrupacion_tomas(matriz_tomas, MAX_TOMAS_POR_DIA)
```

```
agrupacion_final = optimizar_postprocesado(deepcopy(agrupacion_inicial), MAX_TOMAS_POR_DIA)
 costo = costo_total(agrupacion_final)
 # Mostrar resultado
 print("Planificación de grabación:")
 print("----")
 for i, grupo in enumerate(agrupacion_final):
     if grupo: # omitir días vacíos
         actores = actores_para(grupo)
         print(f"Dia \{i+1:2d\}: Tomas \{sorted(grupo)\} \rightarrow Actores \ requeridos: \{len(actores)\}")
 print("\nResumen:")
 print("----")
 print(f"Total de días de grabación : {len([g for g in agrupacion_final if g])}")
 print(f"Total de desplazamientos de actores: {costo}")
Planificación de grabación:
Día 4: Tomas [16, 18, 22] -> Actores requeridos: 2
Día 7: Tomas [17, 23] -> Actores requeridos: 2
Día 10: Tomas [27, 29] -> Actores requeridos: 2
Día 15: Tomas [4, 5, 7, 8, 20, 28] -> Actores requeridos: 6
Día 20: Tomas [2, 3, 10, 14] -> Actores requeridos: 6
Día 24: Tomas [1, 12, 19, 25, 26] -> Actores requeridos: 5
Día 28: Tomas [0, 6, 15, 21, 24] -> Actores requeridos: 6
Día 30: Tomas [9, 11, 13] -> Actores requeridos: 6
```

Resumen:

Total de días de grabación : 8 Total de desplazamientos de actores: 35

### 2.4 (\*)Calcula la complejidad del algoritmo

Respuesta

El cáluclo es de orden de complejidad polinomial

6.Orden polinomial (a > 2), 
$$\mathcal{O}(n^a)$$

Donde  $a \leq 6$ 

El paso dominante es el Greedy inicial, que genera combinaciones hasta 6

Recorre todas las combinaciones posibles de r tomas dentro del conjunto restante.

En el peor caso, genera combinaciones de hasta C(n, r) con r entre 1 y 6.

Por cada combinación, calcula los actores involucrados (costo de O(k) donde k=10 actores máx).

Por tanto, este paso tiene una complejidad de:

$$\mathcal{O}\left(n^6 \cdot a\right)$$

donde:

- n = número de tomas (30)
- a = número de actores (10) constante

Postprocesamiento local (intercambio de tomas entre días):

En el peor caso, compara cada par de días i, j  $(O(d^2))$ . Por cada par, recorre cada toma de un día y evalúa moverla. or cada intento, recalcula actores (máximo 10 por día). En la práctica, con d = n / 6 días, esto es:

$$\mathcal{O}(d^2 \cdot t \cdot a) = \mathcal{O}\left(\left(rac{n}{6}
ight)^2 \cdot 6 \cdot a
ight) = \mathcal{O}(n^2)$$

Criterio	Algoritmo orden Factorial	Algoritmo orden Polinomial
Tipo de solución	Óptima global	Aproximada (no garantiza óptimo)
Estrategia	Backtracking exhaustivo con poda	Selección local de mínimo por día

Criterio	Algoritmo orden Factorial	Algoritmo orden Polinomial		
Complejidad	Factorial ( O(n!) )	orden polinomial ( (a>2), O(n^a) )		
Tiempo de ejecución (30 tomas)	Lento (puede tardar varios minutos)	Rápido (ejecuta en menos de 1 segundo)		
Resultado: desplazamientos actores	<b>29</b> (óptimo)	Mayor (por ejemplo, 31, 34, etc.)		

## 2.5 Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios

Respuesta

Se crea una matriz binaria aleatoria que represente la participación de actores en distintas tomas de una película, que servirá como entrada del algoritmo que agrupa las tomas minimizando el número de actores por día.

Los parámetros configurables, por ejemplo: | Parámetro | Descripción | Valor por defecto | | ------ | ------- | ------- | `NUM\_TOMAS` | Número total de tomas a grabar | 35 | | `NUM\_ACTORES` | Número total de actores disponibles | 12 | | `MAX\_ACTORES\_POR\_TOMA` | Máximo de actores que pueden estar en una toma | 5 | | `MAX\_TOMAS\_DIA` | Máximo de tomas por día | 6 | Estructura de datos generada:

Estructura de datos generada.

Se genera una matriz binaria de tamaño NUM\_TOMAS × NUM\_ACTORES. Cada celda matriz[i][j] representa:

- 1 si el actor j participa en la toma i.
- 0 si el actor j no participa en la toma i

#### Algoritmo de generación (pseudocódigo):

Para cada toma i desde 0 hasta NUM\_TOMAS - 1:
Inicializar toma como lista de ceros de tamaño NUM\_ACTORES
Elegir aleatoriamente entre 1 y MAX\_ACTORES\_POR\_TOMA actores únicos
Para cada actor seleccionado:
toma[actor] = 1
Añadir toma a la matriz

#### Ejemplo de salida con 5 tomas y 4 actores:

Toma 0: [1, 0, 1, 0] # actor 0 y 2 Toma 1: [0, 1, 1, 0] # actor 1 y 2 Toma 2: [0, 0, 1, 1] # actor 2 y 3 Toma 3: [1, 0, 0, 0] # actor 0 Toma 4: [0, 1, 0, 1] # actor 1 y 3

#### Resultado generado:

Una lista de listas (o matriz) que luego se utiliza como input para el algoritmo que realiza la planificación de días.

La idea es que este juego de datos pase como parámetro al algoritmo, que buscará:

- Agrupar las tomas en días (máximo 6 por día),
- Minimizar la cantidad de actores distintos por día.

## 3.1 Aplica el algoritmo al juego de datos generado

Para probar el algoritmo, con un set de datos aleatorios se adapto el programa para que sea parametrizar y evaluar su optimización en tiempo y complejidad, el cual se probó con los siguientes valores:

#### Parámetros configurables:

```
NUM_TOMAS = 35

NUM_ACTORES = 12

MAX_ACTORES_POR_TOMA = 5

MAX_TOMAS_DIA = 6
```

```
In [20]: import random
         from itertools import combinations
         from copy import deepcopy
         import pandas as pd
         # Generación aleatoria del dataset
         def generar_matriz_tomas(n_tomas, n_actores, max_actores_por_toma):
             matriz = []
             for _ in range(n_tomas):
                 toma = [0] * n_actores
                 actores_en_toma = random.sample(range(n_actores), k=random.randint(1, max_actores_por_toma))
                 for a in actores_en_toma:
                     toma[a] = 1
                 matriz.append(toma)
             return matriz
         # Funciones del algoritmo versión 2-mejorada
         # ------
         def actores_para(grupo, matriz):
             actores = set()
             for toma in grupo:
                 for i, val in enumerate(matriz[toma]):
                     if val == 1:
                         actores.add(i)
             return actores
         def costo_total(agrupacion, matriz):
             return sum(len(actores_para(grupo, matriz)) for grupo in agrupacion)
         def greedy_agrupacion_tomas(matriz, max_tomas_dia):
             tomas_disponibles = set(range(len(matriz)))
             resultado = []
             while tomas_disponibles:
                 mejor_grupo = None
                 mejor_costo = float('inf')
                 for r in range(1, min(max_tomas_dia, len(tomas_disponibles)) + 1):
                     for grupo in combinations(tomas_disponibles, r):
                         actores = actores_para(grupo, matriz)
                         if len(actores) < mejor_costo:</pre>
                             mejor_grupo = grupo
                             mejor_costo = len(actores)
                 resultado.append(list(mejor_grupo))
                 tomas_disponibles -= set(mejor_grupo)
             return resultado
         def optimizar_postprocesado(dias, matriz, max_tomas_dia):
             mejorado = True
             while mejorado:
                 mejorado = False
                 for i in range(len(dias)):
                     for j in range(len(dias)):
                         if i == j or not dias[i]:
                             continue
                         for toma in dias[i]:
                             if len(dias[j]) < max_tomas_dia:</pre>
                                 temp_i = dias[i].copy()
                                 temp_j = dias[j].copy()
                                 temp_i.remove(toma)
                                 temp_j.append(toma)
                                 nuevo_costo = len(actores_para(temp_i, matriz)) + len(actores_para(temp_j, matriz))
```

```
actual_costo = len(actores_para(dias[i], matriz)) + len(actores_para(dias[j], matriz))
                       if nuevo_costo < actual_costo:</pre>
                          dias[i] = temp_i
                          dias[j] = temp_j
                          mejorado = True
               if mejorado:
                  break
           if mejorado:
               break
   return dias
# Ejecutar con dataset aleatorio
# Parámetros configurables
NUM_TOMAS = 35
NUM ACTORES = 12
MAX_ACTORES_POR_TOMA = 5
MAX_TOMAS_DIA = 6
# Generar datos aleatorios
random.seed(42)
matriz_tomas = generar_matriz_tomas(NUM_TOMAS, NUM_ACTORES, MAX_ACTORES_POR_TOMA)
# Mostrar la matriz generada
df_matriz = pd.DataFrame(matriz_tomas, columns=[f"Actor {i+1}" for i in range(NUM_ACTORES)])
df_matriz.index = [f"Toma {i+1}" for i in range(NUM_TOMAS)]
print(" | Matriz aleatoria generada (Tomas vs Actores):")
print(df_matriz)
print("\n")
# Aplicar algoritmo
agrupacion_inicial = greedy_agrupacion_tomas(matriz_tomas, MAX_TOMAS_DIA)
agrupacion_final = optimizar_postprocesado(deepcopy(agrupacion_inicial), matriz_tomas, MAX_TOMAS_DIA)
costo = costo_total(agrupacion_final, matriz_tomas)
# Mostrar resultados
print("≦ Planificación de grabación (dataset aleatorio):")
print("----")
for i, grupo in enumerate(agrupacion_final):
   if grupo:
       actores = actores_para(grupo, matriz_tomas)
       print(f"Día {i+1:2d}: Tomas {sorted(grupo)} -> Actores requeridos: {len(actores)}")
print("----")
print(f"Total de días de grabación : {len([g for g in agrupacion_final if g])}")
print(f"Total de desplazamientos de actores: {costo}")
```

<pre>Matriz</pre>	aleator	ia genera	ıda (Tomas	vs Actores	5):			
	Actor 1	Actor 2				Actor 6	Actor 7	\
Toma 1	1	0	0	0	0	0	0	
Toma 2	0	0	1	1	0	0	0	
Toma 3	0	0	0	0	0	0	0	
Toma 4	1	1	0	0	0	0	1	
Toma 5 Toma 6	0	0 0	0 0	1 0	0	0 0	0	
Toma 7	0	0	0	0	0 0	0	0 0	
Toma 8	0	0	0	0	0	0	0	
Toma 9	1	0	0	1	1	0	1	
Toma 10	0	0	0	0	0	0	1	
Toma 11	0	0	1	1	1	0	0	
Toma 12	0	1	0	0	0	0	1	
Toma 13	0	0	0	0	0	1	0	
Toma 14 Toma 15	1	0 1	0 0	0 0	1 0	0 0	0 1	
Toma 16	0	0	0	1	1	1	0	
Toma 17	1	0	0	0	0	0	0	
Toma 18	0	1	0	0	1	0	0	
Toma 19	0	1	0	0	0	0	1	
Toma 20	0	0	0	0	0	1	0	
Toma 21	0	0	0	0	0	1	0	
Toma 22	0	0	0	0	1	0	0	
Toma 23	0	0	0	0	0	0	0	
Toma 24 Toma 25	0	0 0	0 0	1 0	0	0 0	0 1	
Toma 26	0	0	0	1	0 0	0	0	
Toma 27	1	0	0	1	0	0	0	
Toma 28	0	1	0	0	1	0	1	
Toma 29	0	0	0	0	0	1	0	
Toma 30	0	0	0	0	0	0	0	
Toma 31	0	0	1	0	1	0	0	
Toma 32	0	0	0	1	0	0	0	
Toma 33	0	0	0	0	1	1	1	
Toma 34	0 1	0	1	0	0	0 0	0 0	
IOMA 35								
Toma 35	1	1	1	0	0	ð	· ·	
	Actor 8	Actor 9	Actor 10	Actor 11	Actor 12		· ·	
Toma 1					Actor 12	2	Ü	
Toma 1 Toma 2	Actor 8 0 0	Actor 9 0 0	Actor 10 0 0	Actor 11 0 0	Actor 12	<u>2</u> )	· ·	
Toma 1 Toma 2 Toma 3	Actor 8 0 0	Actor 9 0 0 0	Actor 10 0 0 0	Actor 11 0 0 1	Actor 12	<u>2</u> ) L	Ü	
Toma 1 Toma 2 Toma 3 Toma 4	Actor 8 0 0 0	Actor 9 0 0 0 0 1	Actor 10 0 0 0 0	Actor 11 0 0 1	Actor 12	2	Ü	
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5	Actor 8 0 0 0 0	Actor 9 0 0 0 1 0	Actor 10 0 0 0 1	Actor 11 0 0 1 0	Actor 12 6	22	· ·	
Toma 1 Toma 2 Toma 3 Toma 4	Actor 8 0 0 0	Actor 9 0 0 0 0 1	Actor 10 0 0 0 0	Actor 11 0 0 1	Actor 12	2 2 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6	Actor 8 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 0 1	Actor 10 0 0 0 1 0	Actor 11 0 0 1 0 0	Actor 12 6 6 6 6 6 6 6	22		
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 0 1 1	Actor 10 0 0 0 1 0 1 0 0	Actor 11 0 0 1 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	22		
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10	Actor 8 0 0 0 0 0 0 0 0 0 1 0 0	Actor 9 0 0 0 1 1 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 1 0 0 0 0 0 1	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	22		
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11	Actor 8 0 0 0 0 0 0 0 0 1 0 0 0 0	Actor 9 0 0 0 1 1 1 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	2 2 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3		
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12	Actor 8 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14	Actor 8 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0	Actor 11 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0	Actor 11 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 19 Toma 20 Toma 21	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 19 Toma 20 Toma 21 Toma 21 Toma 22 Toma 21 Toma 22	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 19 Toma 20 Toma 21	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1	Actor 10 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26 Toma 27	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0	Actor 10 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26 Toma 27 Toma 28	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 10 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26 Toma 27 Toma 28 Toma 28 Toma 29	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26 Toma 27 Toma 28 Toma 29 Toma 29 Toma 30	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 25 Toma 26 Toma 27 Toma 28 Toma 27 Toma 28 Toma 29 Toma 30 Toma 30 Toma 31	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 26 Toma 27 Toma 28 Toma 29 Toma 29 Toma 30	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 25 Toma 26 Toma 27 Toma 28 Toma 27 Toma 28 Toma 29 Toma 29 Toma 29 Toma 30 Toma 31 Toma 31	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			
Toma 1 Toma 2 Toma 3 Toma 4 Toma 5 Toma 6 Toma 7 Toma 8 Toma 9 Toma 10 Toma 11 Toma 12 Toma 13 Toma 14 Toma 15 Toma 16 Toma 17 Toma 18 Toma 19 Toma 20 Toma 21 Toma 22 Toma 23 Toma 24 Toma 25 Toma 25 Toma 26 Toma 27 Toma 28 Toma 29 Toma 29 Toma 29 Toma 29 Toma 27 Toma 28 Toma 29 Toma 30 Toma 31 Toma 32 Toma 33	Actor 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Actor 9 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0	Actor 10 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Actor 11 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1	Actor 12 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6			

Planificación de grabación (dataset aleatorio):

Día 6: Tomas [0, 16] -> Actores requeridos: 1

Día 8: Tomas [5, 6, 22, 33] -> Actores requeridos: 3

## 3.2 Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo

#### Respuesta

• Para el cálculo de la preguna 1.1, referente a los números de Bell y su fórmula, la referencia que respalda esta en el siguiente enlace:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

https://es.wikipedia.org/wiki/N%C3%BAmero\_de\_Bell

• Algoritmo de Greedy, para la realización de la pregunta 2.3

https://www.geeksforgeeks.org/dsa/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

# 3.3 Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño

#### Respuesta:

Para avanzar en el estudio del problema de planificación de tomas con actores, se pueden explorar en los siguientes puntos:

- 1. La escalabilidad con heurísticas, implementar técnicas como recocido simulado, algoritmos genéticos o búsqueda tabú que permitan abordar instancias mayores (más de 100 tomas y actores) manteniendo tiempos razonables, evitando la explosión combinatoria del enfoque por fuerza bruta.
- 2. Variaciones del problema, se puede incluir restricciones adicionales como:
- Disponibilidad limitada por actor (solo ciertos días).
- Costes diferenciados por actor o por día.
- Tomas que deben grabarse en un orden específico (dependencias).
- 3. Paralelización y optimización híbrida, es decir; dividir el problema en subgrupos o clústeres de tomas similares y resolver localmente, luego optimizar globalmente.
- 4. Estudio probabilístico del espacio de soluciones, analizar cómo cambia la cantidad de combinaciones óptimas al aumentar el número de tomas y actores, lo cual ayuda a estimar la complejidad y ajustar la estrategia algorítmica.

Estos enfoques combinados permiten robustecer la solución frente a escenarios reales y más complejos del mundo del doblaje y producción.