Algoritmos de optimización - Trabajo Práctico

Nombre y Apellidos: **Guillermo Rios Gómez**
Url: https://github.com/GuiRiGo88/03MIAR---Algoritmos-de-Optimizacion---2023/tree/main/TrabajoPractico
Google Colab: https://colab.research.google.com/github/GuiRiGo88/03MIAR---Algoritmos-de-Optimizacion---2023/blob/main/TrabajoPractico?hl=es
Problema:

> 1. **Sesiones de doblaje**

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/36D8IuK
- 1: indica que el actor participa en la toma
- 0: en caso contrario

In [1]:
```
# IMPORTAR LIBRERIA Y DECLARACIÓN DE VARIABLES
import numpy as np

# Datos
num_actores = 10
num_tomas = 30
actores_tomas = [
    [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]
]
```

Modelo

- ¿Como represento el espacio de soluciones?
- ¿Cual es la función objetivo?
- ¿Como implemento las restricciones?

¿Como represento el espacio de soluciones?

Para diseñar el espacio de soluciones (S), donde cada elemento (s) en (S) representa una sesión, y cada sesión (s) contiene un conjunto de tomas (Ts), se sigue la siguiente estructura:

($S = [s1, s2, s3, ..., sn]$)

Donde:

- (n) es el número total de sesiones planificadas.
- Cada (si) se define como una tupla que contiene dos elementos:
- La lista de índices de tomas seleccionadas.
- La lista de actores asociados con esas tomas.

Por lo tanto, cada (si) se representa de la siguiente manera:

($si = ([t1, t2, t3, ..., tm], [a1, a2, a3, ..., ap])$)

Donde:

- ([$t1$, $t2$, $t3$, ..., tm]) representa las tomas seleccionadas para la sesión (i).
- ([$a1$, $a2$, $a3$, ..., ap]) representa los actores asociados con esas tomas.

¿Cual es la función objetivo?

La función objetivo en este caso sería minimizar el costo total de todas las sesiones planificadas, representado por (f(x)).

$minimizar (f(x))$

In [2]:
```
# Función para calcular el costo a minimizar
def costo_total(sesiones_planificadas):
    return sum(costo_total_sesion(sesion) for sesion in sesiones_planificadas)

# Claculo de coste de la solución
def costo_total_sesion(sesion):
    actores_en_sesion = []
    for toma_index in sesion:  # Itera sobre los índices de las tomas
        toma = actores_tomas[toma_index]  # Obtiene la toma correspondiente a ese í
        for actor in range(num_actores):
            if toma[actor] == 1 and actor not in actores_en_sesion:
                actores_en_sesion.append(actor)
    return len(actores_en_sesion)
```

¿Como implemento las restricciones?

- **Restricción de disponibilidad de tomas:** En la función `seleccionar_mejor_toma()`, se asegura de que solo se seleccionen tomas que estén disponibles (no asignadas previamente a otra sesión), ya que se elimina la toma de la lista tomas_disponibles después de asignarla a una sesión.
- **Restricción de actores únicos por sesión:** En la función `costo_total_sesion()`, se garantiza que solo se cuenten los actores únicos presentes en una sesión. Esto se logra al mantener una lista de los actores que ya han sido asignados a la sesión y verificar si un actor aún no está en esa lista antes de agregarlo.
- **Restricción de cantidad máxima de tomas por sesión:** En la función `planificar_sesiones()`, se asegura de que cada sesión contenga como máximo 6 tomas. Esto se logra con un bucle interno en el que se agrega una toma a la sesión actual solo si la longitud de la sesión actual es menor que 6.

In [3]:
```
def seleccionar_mejor_toma(sesiones_asignadas, tomas_disponibles):
    mejor_toma = None
    mejor_costo = float('inf')
    for toma in tomas_disponibles:
        costo = costo_total_sesion(sesiones_asignadas + [toma])
```

```
        if costo < mejor_costo:
            mejor_costo = costo
            mejor_toma = toma
    return mejor_toma

def planificar_sesiones():
    sesiones = []
    actores_por_sesion = []  # Lista para almacenar los actores por sesión
    tomas_disponibles = list(range(num_tomas))

    while tomas_disponibles:
        sesion_actual = []
        actores_en_sesion = set()  # Conjunto para almacenar los actores únicos en
        while len(sesion_actual) < 6 and tomas_disponibles:
            mejor_toma = seleccionar_mejor_toma(sesion_actual, tomas_disponibles)
            sesion_actual.append(mejor_toma)
            tomas_disponibles.remove(mejor_toma)
            # Agregar los actores de la toma actual a la lista de actores en la ses
            for actor, presente in enumerate(actores_tomas[mejor_toma]):
                if presente == 1:
                    actores_en_sesion.add(actor)
        sesiones.append(sesion_actual)
        actores_por_sesion.append(list(actores_en_sesion))  # Convertir el conjunto
    return sesiones, actores_por_sesion  # Devolver las sesiones y la lista de acto
```

¿Que complejidad tiene el problema?. Orden de complejidad y Contabilizar el espacio de soluciones

Dado que el algoritmo completo implica ejecutar `planificar_sesiones()` y luego calcular el costo total, la complejidad total del algoritmo estaría dominada por la complejidad de `planificar_sesiones()`, que es aproximadamente $(O(t \cdot m \cdot n))$, donde (t) es el número total de tomas disponibles.

Donde:

- (t): Es el número total de tomas disponibles.
- (m): Es el número máximo de tomas por sesión.
- (n): Es el número total de sesiones planificadas.

¿Que técnica utilizo? ¿Por qué?

Técnica Utilizada: Optimización Voraz

Razones de Utilización:

- **Selección Local Óptima**: En cada paso del algoritmo, se toma una decisión localmente óptima, es decir, se selecciona la toma que proporciona el menor costo adicional para la sesión actual. Esta estrategia se alinea con la naturaleza de los algoritmos voraces, donde las decisiones se toman considerando solo la información disponible en el

momento.

- **Decisiones Basadas en Información Local**: El algoritmo toma decisiones basadas únicamente en la información local, sin considerar el impacto global en todas las sesiones futuras. Selecciona la mejor toma disponible en cada paso, sin revisar decisiones anteriores. Este enfoque es característico de los algoritmos voraces.
- **Estructura de los Datos y Naturaleza del Problema**: Dado que el problema implica minimizar el costo total de las sesiones y las tomas tienen diferentes costos y actores asociados, una estrategia voraz es apropiada. La estructura del problema permite abordarlo de manera efectiva seleccionando tomas de manera voraz para cada sesión sin necesidad de explorar todas las combinaciones posibles.
- **Eficiencia de Tiempo y Espacio**: Aunque la búsqueda exhaustiva garantizaría la solución óptima, podría ser prohibitivamente costosa en términos de tiempo y recursos computacionales debido al tamaño del espacio de búsqueda. En cambio, la estrategia voraz ofrece una solución razonablemente buena en un tiempo de ejecución más eficiente.

In [4]:
```
def imprimir_planificacion_con_costo_total():
    sesiones_planificadas, actores_por_sesion = planificar_sesiones()  # Obtener se
    imprimir_planificacion(sesiones_planificadas, actores_por_sesion)
    print(f"\nCoste objetivo a Minimizar: {sum(sum(fila) for fila in actores_tomas)
    print(f"Costo final Minimizado: {costo_total(sesiones_planificadas)}")


def imprimir_planificacion(sesiones, actores_por_sesion):
    for i, (sesion, actores) in enumerate(zip(sesiones, actores_por_sesion), start=
        print(f"Sesión {i}: {sesion}, Actores: {actores}, Costo Total: {costo_total

# Ejecutar el planificador y mostrar resultados
sesiones_planificadas = planificar_sesiones()
imprimir_planificacion_con_costo_total()
```

```
Sesión 1: [15, 26, 1, 12, 16, 18], Actores: [0, 2, 3, 4, 9], Costo Total: 5
Sesión 2: [17, 23, 13, 22, 7, 8], Actores: [0, 1, 2, 3, 5], Costo Total: 5
Sesión 3: [20, 4, 27, 29, 3, 14], Actores: [0, 1, 3, 5, 6, 7], Costo Total: 6
Sesión 4: [2, 5, 6, 0, 19, 21], Actores: [0, 1, 2, 3, 4, 6], Costo Total: 6
Sesión 5: [28, 9, 25, 10, 11, 24], Actores: [0, 1, 2, 3, 4, 5, 7, 8, 9], Costo Total
: 9

Coste objetivo a Minimizar: 94
Costo final Minimizado: 31
```