Nombre: Amaia Navarrete Muro

https://colab.research.google.com/drive/1wPcamjf4gOu9REDKUKTpUNtYmri3IiGe?usp=sharing

https://github.com/anavarrete9/03MIAR---Algoritmos-de-Optimizacion-2022-2023

```
#DIVIDE Y VENCERÁS
#Torres de Hanoi
def Torres_Hanoi(N, desde, hasta):
  #N = número de fichas
  #desde = número de poste
  #hasta = número de poste de llegada

  if N == 1:
    print('Lleva la ficha desde ' + str(desde) + ' hasta ' + str(hasta))
  else:
    Torres_Hanoi(N-1, desde, 6-desde-hasta)
    print('Lleva la ficha desde ' + str(desde) + ' hasta ' + str(hasta))
    Torres_Hanoi(N-1, 6-desde-hasta, hasta)


Torres_Hanoi(4,1,3)
```

```
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 3 hasta 1
Lleva la ficha desde 3 hasta 2
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 2 hasta 1
Lleva la ficha desde 3 hasta 1
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
```

```
#TÉCNICA VORAZ
def cambio_monedas(cantidad, sistema):
  solucion = []
  acumulado = 0
  for moneda in sistema:
    n_monedas = (cantidad - acumulado) // moneda
    solucion.append(n_monedas)
    acumulado += n_monedas * moneda
    if acumulado == cantidad:
      return solucion

#Función clase
def cambio_monedas_clase(cantidad, sistema):
  # cantidad a cambiar
  # sistema: valor de las monedas diferente, ordenado de mayor valor a menor
  solucion = {}
  for v in sistema:
    monedas = cantidad // v
    solucion[v] = monedas
    cantidad -= monedas * v
    if cantidad == 0:
      return solucion


print(cambio_monedas(123, [25,10,5,1]))
print(cambio_monedas_clase(123, [25,10,5,1]))
```

```
[4, 2, 0, 3]
{25: 4, 10: 2, 5: 0, 1: 3}
```

```
#TÉCNICA DE VUELTRA ATRÁS. BACKTRACKING
#Problema de las n reinas

#Función auxiliar para ver si la solución es factible
def es_prometedora(solucion, etapa):
  for i in range(etapa + 1):
    if solucion.count(solucion[i]) >1:
      return False
    for j in range(i+1, etapa +1):
      if abs(i-j) == abs(solucion[i] - solucion[j]):
        return False
```

```
    return True

#Visualizar la solución
def escribe_solucion(s):
  n = len(s)
  for x in range(n):
    print('')
    for i in range(n):
      if s[i] == x+1:
        print(' x ', end='')
      else:
        print(' - ', end='')

#Función principal
def reinas(N, sol = [], etapa = 0):
  #N: tamaño del tablero y número de reinas
  #sol: solución parcial
  #etapa: número de reinas colocadas en la solución parcial
  if len(sol) == 0:
    sol = [0 for i in range(N)]

  for i in range(1, N+1):
    sol[etapa] = i

    if es_prometedora(sol, etapa):
      if etapa == N-1:
        print('\n\nLa solución es:')
        print(sol)
        escribe_solucion(sol)
      else:
        reinas(N, sol, etapa+1)

    else:
      None
    sol[etapa] = 0

reinas(4)
```

```
 La solución es:
 [2, 4, 1, 3]

  -  -  x  -
  x  -  -  -
  -  -  -  x
  -  x  -  -

 La solución es:
 [3, 1, 4, 2]

  -  x  -  -
  -  -  -  x
  x  -  -  -
  -  -  x  -
```