Actividad Guiada 1 de Algoritmos de Optimizacion

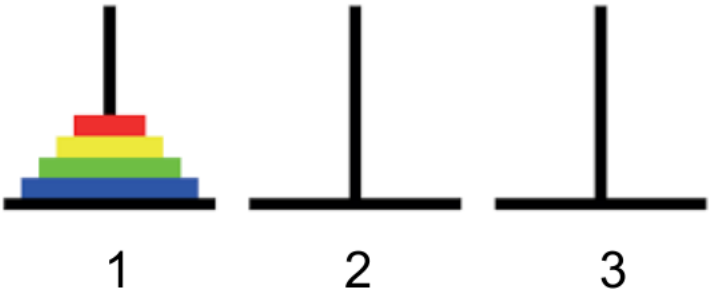
Nombre: Victor Garcia Alcalde

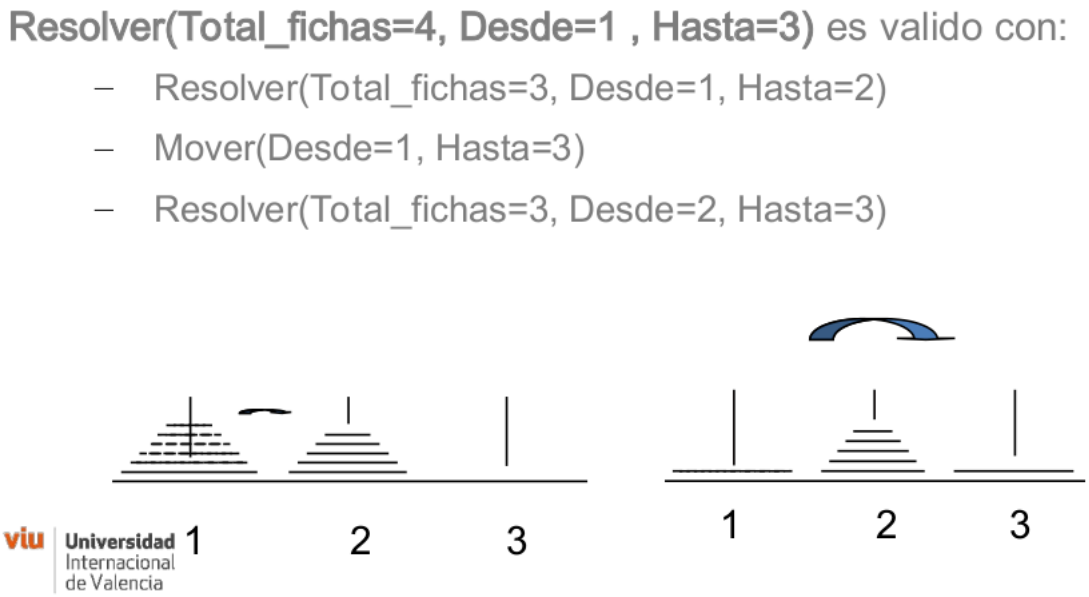
https://colab.research.google.com/github/victorgarciaalcalde-ctrl/03MIAR-AlgoritmosOptimizacion/blob/main/Algoritmos_AG1.ipynb

https://github.com/victorgarciaalcalde-ctrl/03MIAR-AlgoritmosOptimizacion/tree/main

1. Divide y vencerás - Torres de Hanoi
2. Algoritmo Voraz - Cambio de monedas
3. Backtraining - N reinas

Torres de Hanoi - Divide y venceras





El problema de las Torres de Hanoi consiste en mover un conjunto de discos de diferentes tamaños desde una torre inicial (1) a una torre final (3), para ayudarte existe una torre auxiliar (3).

Las reglas son:

- Solo se puede mover un disco cada vez
- Un disco grande no se podran colocar sobre discos pequeños

El porblema se resuelve utilizando la tecnica de divide y venceras, seguiremos los siguientes pasos:

1. Se moveran N-1 discos superiores a la torre auxiliar
2. Se movera el disco mas grande desde la torre inicial a la torre final

```
#Torres de Hanoi - Divide y venceras
################################################################################


#################################################################
def Torres_Hanoi(N, desde, hasta):
  #N - Nº de fichas
  #desde - torre inicial
  #hasta - torre fina
  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(5, 1, 3)
#################################################################
```

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

Cambio de monedas - Técnica voraz

En este ejercicio se utiliza un algoritmo voraz para resolver el problema del cambio de monedas.

La estrategia voraz consiste en seleccionar en cada paso la moneda de mayor valor posible que no supere la cantidad restante por devolver. De este modo, se va reduciendo progresivamente la cantidad hasta alcanzar el valor objetivo.

Este enfoque no garantiza siempre la solución óptima para cualquier sistema de monedas, pero funciona correctamente cuando el sistema está bien diseñado, como en este caso.

```
#Cambio de monedas - Técnica voraz
################################################################################
SISTEMA = [11, 5 ,1  ]
#################################################################
def cambio_monedas(CANTIDAD,SISTEMA):
#....
  SOLUCION = [0]*len(SISTEMA)
  ValorAcumulado = 0

  for i,valor in enumerate(SISTEMA):
    monedas = (CANTIDAD-ValorAcumulado)//valor
    SOLUCION[i] = monedas
    ValorAcumulado = ValorAcumulado + monedas*valor

    if CANTIDAD == ValorAcumulado:
      return SOLUCION

  print("No es posible encontrar solucion")
cambio_monedas(15,SISTEMA)
```

```
##################################################################
```

```
[1, 0, 4]
```

```
Empieza a programar o a crear código con IA.
```

N Reinas - Vuelta Atrás(Backtracking)

En este ejercicio se resuelve el problema de las N reinas mediante la técnica de vuelta atrás (backtracking).

La idea consiste en construir una solución por etapas, colocando una reina en cada fila. En cada paso se comprueba si la colocación actual es prometedora, es decir, si no entra en conflicto con las reinas ya colocadas (misma columna o diagonales).

Si una colocación no es válida, se retrocede (backtrack) y se prueba la siguiente alternativa. El proceso continúa hasta encontrar una solución completa o agotar todas las posibilidades.

```
#N Reinas - Vuelta Atrás()
##############################################################################

#Verifica que en la solución parcial no hay amenzas entre reinas
####################################################################
def es_prometedora(SOLUCION,etapa):
####################################################################
  #print(SOLUCION)
  #Si la solución tiene dos valores iguales no es valida => Dos reinas en la misma fila
  for i in range(etapa+1):
    #print("El valor " + str(SOLUCION[i]) + " está " +  str(SOLUCION.count(SOLUCION[i])) + " veces")
    if SOLUCION.count(SOLUCION[i]) > 1:
      return False

    #Verifica las diagonales
    for j in range(i+1, etapa +1 ):
      #print("Comprobando diagonal de " + str(i) + " y " + str(j))
      if abs(i-j) == abs(SOLUCION[i]-SOLUCION[j]) : return False
  return True

#Traduce la solución al tablero
####################################################################
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="")



#Proceso principal de N-Reinas
####################################################################
def reinas(N, solucion=[],etapa=0):
####################################################################
### ....
  if len(solucion) == 0:         # [0,0,0...]
    solucion = [0 for i in range(N) ]
```

```
  for i in range(1, N+1):
    solucion[etapa] = i
    if es_prometedora(solucion, etapa):
      if etapa == N-1:
        print(solucion)
      else:
        reinas(N, solucion, etapa+1)
    else:
      None

  solucion[etapa] = 0

reinas(5,solucion=[],etapa=0)
```

```
[1, 3, 5, 2, 4]
[1, 4, 2, 5, 3]
[2, 4, 1, 3, 5]
[2, 5, 3, 1, 4]
[3, 1, 4, 2, 5]
[3, 5, 2, 4, 1]
[4, 1, 3, 5, 2]
[4, 2, 5, 3, 1]
[5, 2, 4, 1, 3]
[5, 3, 1, 4, 2]
```

```
escribe_solucion([1, 5, 8, 6, 3, 7, 2, 4])
```

```
 X  -  -  -  -  -  -  -
 -  -  -  -  -  -  X  -
 -  -  -  -  X  -  -  -
 -  -  -  -  -  -  -  X
 -  X  -  -  -  -  -  -
 -  -  -  X  -  -  -  -
 -  -  -  -  -  X  -  -
 -  -  X  -  -  -  -  -
```