Actividad Guiada 1 de Algoritmos de Optimización

Nombre: José Jesús La Casa Nieto

https://colab.research.google.com/drive/1dEq5PFVTicwJ6ska-QXuCuaboI2nmV6Z?usp=sharing

https://github.com/JoseJesusLaCasaNieto/03MIAR---Algoritmos-de-Optimizacion---2024

Torres de Hanoi - Divide y vencerás

```
# Torres de Hanoi - Divide y vencerás

def Torres_Hanoi(N, desde, hasta):
  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(6, 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
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 3 hasta 1
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 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 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
Lleva la ficha desde 2 hasta 1
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 3 hasta 1
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
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
```

Cambio de monedas - Técnica voraz

```python
# Cambio de monedas - Técnica voraz

SISTEMA = [25, 10, 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 solución")

cambio_monedas(37, SISTEMA)
```

```
[1, 1, 0, 2]
```

N Reinas - Vuelta atrás

```
# N Reinas - Vuelta atrás

# Verifica que en la solución parcial no hay amenazas entre reinas
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

# 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:
    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(8, solucion=[], etapa=0)
```

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

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

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

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

Viaje por el río - Programación dinámica

```python
# Viaje por el río - Programación dinámica

TARIFAS = [
    [0, 5, 4, 3, 999, 999, 999],
    [999, 0, 999, 2, 3, 999, 11],
    [999, 999, 0, 1, 999, 4, 10],
    [999, 999, 999, 0, 5, 6, 9],
    [999, 999, 999, 999, 0, 999, 4],
    [999, 999, 999, 999, 999, 0, 3],
    [999, 999, 999, 999, 999, 999, 0]
]

# Cálculo de la matriz PRECIOS y RUTAS
def Precios(TARIFAS):
  N = len(TARIFAS[0])
  PRECIOS = [[9999] * N for i in [9999] * N]
  RUTA = [[""] * N for i in [""] * N]

  for i in range(0, N-1):
    RUTA[i][i] = i
    PRECIOS[i][i] = 0
    for j in range(i+1, N):
      MIN = TARIFAS[i][j]
      RUTA[i][j] = i
      for k in range(i, j):
        if PRECIOS[i][k] + TARIFAS[k][j] < MIN:
          MIN = min(MIN, PRECIOS[i][k] + TARIFAS[k][j])
          RUTA[i][j] = k
        PRECIOS[i][j] = MIN

  return PRECIOS, RUTA

PRECIOS, RUTA = Precios(TARIFAS)

print("PRECIOS")
for i in range(len(TARIFAS)):
  print(PRECIOS[i])
```

```
print("\nRUTA")
for i in range(len(TARIFAS)):
  print(RUTA[i])

def calcular_ruta(RUTA, desde, hasta):
  if desde == hasta:
    return ""
  else:
    return str(calcular_ruta(RUTA, desde, RUTA[desde][hasta])) + ',' +
str(RUTA[desde][hasta])

print("\nLa ruta es:")
calcular_ruta(RUTA, 0, 6)
```

```
PRECIOS
[0, 5, 4, 3, 8, 8, 11]
[9999, 0, 999, 2, 3, 8, 7]
[9999, 9999, 0, 1, 6, 4, 7]
[9999, 9999, 9999, 0, 5, 6, 9]
[9999, 9999, 9999, 9999, 0, 999, 4]
[9999, 9999, 9999, 9999, 9999, 0, 3]
[9999, 9999, 9999, 9999, 9999, 9999, 9999]

RUTA
[0, 0, 0, 0, 1, 2, 5]
['', 1, 1, 1, 1, 3, 4]
['', '', 2, 2, 3, 2, 5]
['', '', '', 3, 3, 3, 3]
['', '', '', '', 4, 4, 4]
['', '', '', '', '', 5, 5]
['', '', '', '', '', '', '']

La ruta es:

{"type":"string"}
```

Práctica individual - Encontrar los dos puntos más cercanos

```
# Lista de números 1D - Fuerza bruta

import random

def numeros_mas_cercanos_fuerza_bruta(lista):
    min_diferencia = float('inf')
    numeros_cercanos = None, None

    for i in range(len(lista)):
        for j in range(i + 1, len(lista)):
            diferencia = abs(lista[i] - lista[j])
            if diferencia < min_diferencia:
                min_diferencia = diferencia
```

```
                numeros_cercanos = lista[i], lista[j]

    return min_diferencia, numeros_cercanos

lst = [random.randrange(1, 10000) for x in range(100)]
distancia_minima, par_puntos = numeros_mas_cercanos_fuerza_bruta(lst)
print("Distancia mínima entre dos puntos:", distancia_minima)
print("Los dos puntos más cercanos son:", par_puntos)
```

```
Distancia mínima entre dos puntos: 2
Los dos puntos más cercanos son: (5491, 5493)
```

El orden de complejidad del algoritmo de fuerza bruta en nuestro caso es de O(n^2).

Este algoritmo se vuelve ineficiente para grandes conjuntos de datos por su complejidad cuadrática. Para mejorarlo, deberíamos usar otros algoritmos.

```
# Lista de números 1D - Divide y vencerás

import math
import random

def distancia(p1, p2):
    return abs(p1 - p2)

def distancia_minima_divide_y_venceras(puntos):
    puntos_ordenados = sorted(puntos)
    return
distancia_minima_divide_y_venceras_recursiva(puntos_ordenados)

def distancia_minima_divide_y_venceras_recursiva(puntos):
    n = len(puntos)
    if n <= 3:
        return distancia_minima_fuerza_bruta(puntos)

    medio = n // 2
    punto_medio = puntos[medio]

    izquierda = puntos[:medio]
    derecha = puntos[medio:]

    distancia_izquierda, par_izquierda =
distancia_minima_divide_y_venceras_recursiva(izquierda)
    distancia_derecha, par_derecha =
distancia_minima_divide_y_venceras_recursiva(derecha)

    distancia_minima = min(distancia_izquierda, distancia_derecha)
    par_minimo = par_izquierda if distancia_minima ==
distancia_izquierda else par_derecha
```

```
    franja_central = [punto for punto in puntos if abs(punto -
punto_medio) < distancia_minima]
    distancia_franja, par_franja =
distancia_minima_franja(franja_central, distancia_minima)

    if distancia_franja < distancia_minima:
        return distancia_franja, par_franja
    else:
        return distancia_minima, par_minimo

def distancia_minima_franja(franja_central, distancia_minima):
    franja_central.sort()
    n = len(franja_central)
    minima_distancia = distancia_minima
    par_minimo = None

    for i in range(n):
        for j in range(i+1, min(i+8, n)):
            if abs(franja_central[j] - franja_central[i]) <
minima_distancia:
                minima_distancia = distancia(franja_central[i],
franja_central[j])
                par_minimo = (franja_central[i], franja_central[j])

    return minima_distancia, par_minimo

def distancia_minima_fuerza_bruta(puntos):
    minima_distancia = float('inf')
    n = len(puntos)
    par_minimo = None

    for i in range(n):
        for j in range(i+1, n):
            if abs(puntos[j] - puntos[i]) < minima_distancia:
                minima_distancia = distancia(puntos[i], puntos[j])
                par_minimo = (puntos[i], puntos[j])

    return minima_distancia, par_minimo

puntos = [random.randrange(1, 100000) for x in range(100)]
distancia_minima, par_puntos =
distancia_minima_divide_y_venceras(puntos)
print("Distancia mínima entre dos puntos:", distancia_minima)
print("Los dos puntos más cercanos son:", par_puntos)
```

```
Distancia mínima entre dos puntos: 9
Los dos puntos más cercanos son: (25453, 25462)
```

El orden de complejidad del algoritmo divide y vencerás es de O(n.log(n)).

Para este caso, hay distintas maneras de mejorar el algoritmo. Por un lado, se podría optimizar la búsqueda en la franja central, utilizando una técnica de vecinos más cercanos. Otra opción, es eliminar los puntos que sean innecesarios de la franja central. Por último, se podrían utilizar librerías externas que realicen el trabajo de forma más óptima.

```
# Lista de números 2D - Divide y vencerás

import math
import random

def distancia(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def distancia_minima_divide_y_venceras(puntos):
    puntos_ordenados_x = sorted(puntos, key=lambda x: x[0])
    return
distancia_minima_divide_y_venceras_recursiva(puntos_ordenados_x)

def distancia_minima_divide_y_venceras_recursiva(puntos):
    n = len(puntos)
    if n <= 3:
        return distancia_minima_fuerza_bruta(puntos)

    medio = n // 2
    punto_medio = puntos[medio]

    izquierda = puntos[:medio]
    derecha = puntos[medio:]

    distancia_izquierda, par_izquierda =
distancia_minima_divide_y_venceras_recursiva(izquierda)
    distancia_derecha, par_derecha =
distancia_minima_divide_y_venceras_recursiva(derecha)

    distancia_minima = min(distancia_izquierda, distancia_derecha)
    par_minimo = par_izquierda if distancia_minima ==
distancia_izquierda else par_derecha

    franja_central = [punto for punto in puntos if abs(punto[0] -
punto_medio[0]) < distancia_minima]
    distancia_franja, par_franja =
distancia_minima_franja(franja_central, distancia_minima)

    if distancia_franja < distancia_minima:
        return distancia_franja, par_franja
    else:
        return distancia_minima, par_minimo

def distancia_minima_franja(franja_central, distancia_minima):
    franja_central.sort(key=lambda x: x[1])
```

```
    n = len(franja_central)
    minima_distancia = distancia_minima
    par_minimo = None

    for i in range(n):
        for j in range(i+1, min(i+8, n)):
            if abs(franja_central[j][1] - franja_central[i][1]) <
minima_distancia:
                dist = distancia(franja_central[i], franja_central[j])
                if dist < minima_distancia:
                    minima_distancia = dist
                    par_minimo = (franja_central[i],
franja_central[j])

    return minima_distancia, par_minimo

def distancia_minima_fuerza_bruta(puntos):
    minima_distancia = float('inf')
    n = len(puntos)
    par_minimo = None

    for i in range(n):
        for j in range(i+1, n):
            dist = distancia(puntos[i], puntos[j])
            if dist < minima_distancia:
                minima_distancia = dist
                par_minimo = (puntos[i], puntos[j])

    return minima_distancia, par_minimo

puntos = [(random.randrange(1, 10000), random.randrange(1, 10000)) for
x in range(100)]
distancia_minima, par_puntos =
distancia_minima_divide_y_venceras(puntos)
print(f"Distancia mínima entre dos puntos: {distancia_minima:.2f}")
print("Los dos puntos más cercanos son:", par_puntos)
```

```
Distancia mínima entre dos puntos: 70.52
Los dos puntos más cercanos son: ((7179, 9299), (7201, 9366))
```

```
# Lista de números 3D - Divide y vencerás

import math
import random

def distancia(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2]
- p2[2])**2)

def distancia_minima_divide_y_venceras(puntos):
    puntos_ordenados_x = sorted(puntos, key=lambda x: x[0])
```

```
    return
distancia_minima_divide_y_venceras_recursiva(puntos_ordenados_x)

def distancia_minima_divide_y_venceras_recursiva(puntos):
    n = len(puntos)
    if n <= 3:
        return distancia_minima_fuerza_bruta(puntos)

    medio = n // 2
    punto_medio = puntos[medio]

    izquierda = puntos[:medio]
    derecha = puntos[medio:]

    distancia_izquierda, par_izquierda =
distancia_minima_divide_y_venceras_recursiva(izquierda)
    distancia_derecha, par_derecha =
distancia_minima_divide_y_venceras_recursiva(derecha)

    distancia_minima = min(distancia_izquierda, distancia_derecha)
    par_minimo = par_izquierda if distancia_minima ==
distancia_izquierda else par_derecha

    franja_central = [punto for punto in puntos if abs(punto[0] -
punto_medio[0]) < distancia_minima]
    distancia_franja, par_franja =
distancia_minima_franja(franja_central, distancia_minima)

    if distancia_franja < distancia_minima:
        return distancia_franja, par_franja
    else:
        return distancia_minima, par_minimo

def distancia_minima_franja(franja_central, distancia_minima):
    franja_central.sort(key=lambda x: x[1])
    n = len(franja_central)
    minima_distancia = distancia_minima
    par_minimo = None

    for i in range(n):
        for j in range(i+1, min(i+8, n)):
            if abs(franja_central[j][1] - franja_central[i][1]) <
minima_distancia:
                dist = distancia(franja_central[i], franja_central[j])
                if dist < minima_distancia:
                    minima_distancia = dist
                    par_minimo = (franja_central[i],
franja_central[j])

    return minima_distancia, par_minimo
```

```python
def distancia_minima_fuerza_bruta(puntos):
    minima_distancia = float('inf')
    n = len(puntos)
    par_minimo = None

    for i in range(n):
        for j in range(i+1, n):
            dist = distancia(puntos[i], puntos[j])
            if dist < minima_distancia:
                minima_distancia = dist
                par_minimo = (puntos[i], puntos[j])

    return minima_distancia, par_minimo

puntos = [(random.randrange(1, 10000), random.randrange(1, 10000),
random.randrange(1, 10000)) for x in range(100)]
distancia_minima, par_puntos =
distancia_minima_divide_y_venceras(puntos)
print(f"Distancia mínima entre dos puntos: {distancia_minima:.2f}")
print("Los dos puntos más cercanos son:", par_puntos)
```

```
Distancia mínima entre dos puntos: 146.40
Los dos puntos más cercanos son: ((3430, 1186, 5541), (3455, 1311,
5613))
```