#Actividad Guiada 1 de Algoritmos de Optimizacion

Nombre: Sergi Ribera Ortells Enlace Google_Colab: https://colab.research.google.com/drive/1W3IoYoVPij_cdLQDeOloqIOzeuzu2ORO#scrollTo=t9vPq_UqH_Hn

Torres de Hanoi - Divide y venceras

El número de la ficha es creciente, la ficha 1 es la más pequeña, la 2 es la segunda más pequeña, y asi progresivamente

```
def Torres_Hanoi(N, desde, hasta):
  #N - Nº de fichas
  #desde - torre inicial
  #hasta - torre fina
  if N==1 :
    print("Lleva la ficha " + str(N) + " desde " + str(desde) + "
hasta " + str(hasta))

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

Torres_Hanoi(4, 1, 3)
```

```
Lleva la ficha 1 desde 1 hasta 2
Lleva la ficha 2 desde 1 hasta 3
Lleva la ficha 1 desde 2 hasta 3
Lleva la ficha 3 desde 1 hasta 2
Lleva la ficha 1 desde 3 hasta 1
Lleva la ficha 2 desde 3 hasta 2
Lleva la ficha 1 desde 1 hasta 2
Lleva la ficha 4 desde 1 hasta 3
Lleva la ficha 1 desde 2 hasta 3
Lleva la ficha 2 desde 2 hasta 1
Lleva la ficha 1 desde 3 hasta 1
Lleva la ficha 3 desde 2 hasta 3
Lleva la ficha 1 desde 1 hasta 2
Lleva la ficha 2 desde 1 hasta 3
Lleva la ficha 1 desde 2 hasta 3
```

#Cambio de monedas - Técnica voraz

```
SISTEMA = [10, 5 ,  1  ]

def cambio_monedas(CANTIDAD,SISTEMA):
```

```
# CANTIDAD es el dinero que se tiene que devolver
# SISTEMA son los importes disponibles para devolver el cambio

  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:
      for j, valores in enumerate(SISTEMA):
        if SOLUCION[j] not in (0, 1):
          print("Se devuelven "+ str(SOLUCION[j]) + " monedas de " +
str(valores) + " centimos")
        if SOLUCION[j] == 1:
          print("Se devuelve "+ str(SOLUCION[j]) + " moneda de " +
str(valores) + " centimos")
      return SOLUCION

  print("No es posible encontrar solucion")

cambio_monedas(128,SISTEMA)
```

```
Se devuelven 12 monedas de 10 centimos
Se devuelve 1 moneda de 5 centimos
Se devuelven 3 monedas de 1 centimos

[12, 1, 3]
```

#N Reinas - Vuelta Atrás

```
#Verifica que en la solución parcial no hay amenazas 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="")
  print("\n")



#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 )

        escribe_solucion(solucion)
      else:
        reinas(N, solucion, etapa+1)
    else:
      None

  solucion[etapa] = 0

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

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

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

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

 X  -  -  -  -
 -  -  X  -  -
 -  -  -  -  X
 -  X  -  -  -
 -  -  -  X  -

[2, 4, 1, 3, 5]

 -  -  X  -  -
 X  -  -  -  -
 -  -  -  X  -
 -  X  -  -  -
 -  -  -  -  X

[2, 5, 3, 1, 4]

 -  -  -  X  -
 X  -  -  -  -
 -  -  X  -  -
 -  -  -  -  X
 -  X  -  -  -

[3, 1, 4, 2, 5]

 -  X  -  -  -
 -  -  -  X  -
 X  -  -  -  -
 -  -  X  -  -
 -  -  -  -  X

[3, 5, 2, 4, 1]

 -  -  -  -  X
 -  -  X  -  -
 X  -  -  -  -
 -  -  -  X  -
 -  X  -  -  -

[4, 1, 3, 5, 2]

 -  X  -  -  -
 -  -  -  -  X
 -  -  X  -  -
 X  -  -  -  -
 -  -  -  X  -

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

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

[5, 2, 4, 1, 3]

 -  -  -  X  -
 -  X  -  -  -
 -  -  -  -  X
 -  -  X  -  -
 X  -  -  -  -

[5, 3, 1, 4, 2]

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

#Viaje por el rio - 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]
]

#999 se puede sustituir por float("inf")


#Calculo de la matriz de PRECIOS y RUTAS

def Precios(TARIFAS):

  #Total de Nodos
  N = len(TARIFAS[0])

  #Inicialización de la tabla de precios
  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              #Para ir de i a i se "pasa por i"
    PRECIOS[i][i] = 0           #Para ir de i a i se se paga 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          #Anota que para ir de i a j hay
que pasar por k
        PRECIOS[i][j] = MIN

  return PRECIOS,RUTA
#####################################################################

PRECIOS,RUTA = Precios(TARIFAS)
#print(PRECIOS[0][6])

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

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

#Determinar la ruta con Recursividad
def calcular_ruta(RUTA, desde, hasta):
  if desde == hasta:
    #print("Ir a :" + str(desde))
    return desde
  else:
    return str(calcular_ruta( RUTA, desde, RUTA[desde][hasta])) +  \
                ',' + \
                str(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"}
```

Problema: Encontrar los dos puntos más cercano • Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos

Primero lo vamos a intentar para 1 dimensión

Fuera bruta:

```
def puntos_cercanos1(vector):
  a = vector[0]
  b = vector[1]
  dist= abs(vector[0] - vector[1])

  for i in range(0,len(vector)-1):
    for j in range(i+1,len(vector)):
      if abs(vector[i] - vector[j]) < dist:
        dist = abs(vector[i] - vector[j])
        a = vector[i]
        b = vector[j]

  print("los puntos mas cercanos son: " + str(a) + " y " + str(b))
  return [a,b]

import random
lista1 = [random.randrange(1, 10000) for x in range(30)]
print(str((lista1)))
puntos_cercanos1(lista1)
```

```
[4546, 792, 1010, 5311, 51, 2865, 3747, 9526, 8100, 2293, 1879, 1850,
9449, 6239, 4518, 6849, 4766, 8981, 2428, 6265, 6709, 4020, 5149,
6657, 5696, 9171, 8399, 6142, 4997, 7962]
los puntos mas cercanos son: 6239 y 6265

[6239, 6265]
```

Esta solución tiene una complejidad de orden cuadratico:

$$O\left(n^2\right)$$

Vamos a probar a aplicar el metodo de divide y vencerás para optimizar este cálculo:

```
def puntos_cercanos2(vector):
  if len(vector) == 2:
    return vector

  else:
    v1 = puntos_cercanos2(vector[:-1])
    v2 = puntos_cercanos2(vector[1:])
    v3 = puntos_cercanos2([vector[0],vector[-1]])
    if abs(v1[1] - v1[0]) <=  abs(v2[1] - v2[0]) and  abs(v1[1] -
v1[0]) <=  abs(v3[1] - v3[0]):
      return v1
    if abs(v2[1] - v2[0]) <=  abs(v1[1] - v1[0]) and  abs(v2[1] -
v2[0]) <=  abs(v3[1] - v3[0]):
      return v2
    else:
      return v3



import random
lista1 = [random.randrange(1, 100) for x in range(10)]

print(str((lista1)))
sol = puntos_cercanos2(lista1)
print("los puntos mas cercanos son: " + str(sol[0]) + " y " +
str(sol[1]))
```

```
[42, 75, 44, 34, 34, 29, 43, 52, 7, 91]
los puntos mas cercanos son: 34 y 34
```

Analizando la complejidad de este algoritmo vemos que cada vez que entra a la función con un vector de dimension n llama dos veces a la fucion con dimensión n-1, asi sucesivamente hasta llevar a dimensión 2. Analizando esto de la misma forma que el el código del problema de las torres de Hanoi, por tanto vemos que no llega a ser más eficiente que por fuerza bruta, si no que aumenta la complejidad a orden exponencial:

$$\boldsymbol{O}\left(2^{n}\right)$$

Esto se podría mejorar si aplicamos la funcion sort al conjunto, que tiene una complejidad de orden

$$\boldsymbol{O}\left(n\log\left(n\right)\right)$$

y en lugar de cada vez acceder de forma recursiva a la función puntos_cercanos2 en dimension n-1 acceder en dos mitades, y comparar cada mitad con la distancia entre los dos puntos centrales tambien. esta función tambien tiene complejidad de orden :

$$O(n \log(n))$$

Por tanto la suma de las complejidades sigue siendo:

$$O(n \log(n))$$

```
def puntos_cercanos3(vector):
  vector.sort()
  if len(vector) == 2:
    return vector

  else:
    n = len(vector)//2
    v1 = puntos_cercanos3(vector[:n+1])
    v2 = puntos_cercanos3(vector[n:])
    if abs(v1[1] - v1[0]) <=  abs(v2[1] - v2[0]) :
      return v1
    else:
      return v2



import random
lista1 = [random.randrange(1, 100) for x in range(10)]

print(str((lista1)))

sol = puntos_cercanos3(lista1)
print("los puntos mas cercanos son: " + str(sol[0]) + " y " +
str(sol[1]))
```

```
[42, 69, 40, 89, 58, 41, 62, 21, 24, 1]
los puntos mas cercanos son: 40 y 41
```

Por último, se puede ampliar este problema a listas con vectores [[2,5],[4,8],...] de dos dimensiones o de más dimensiones si en de calcular la siguiente diferencia:

v1[1] - v1[0]

Creamos una funcion que nos devuelva la distancia euclidea entre dos vectores y la aplicamos a estos dos valores v1[0] y v1[1]:

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 \ldots + (x_n - y_n)^2}$$

para vectores de dimension n.

La función podria ser como la sigueinte:

```
import math

def dist_euclidea(vect):
```

```
   suma_diferencias_cuadradas = sum((v1 - v2)**2 for v1, v2 in
zip(vect[0], vect[1]))
   return math.sqrt(suma_diferencias_cuadradas)
```

Y con esto quedaría terminada la prática 1.

Un saludo profe. 😀