Algoritmos - Actividad Guiada 1

Nombre: Alex Sanz Morant
URL:
https://colab.research.google.com/drive/18CMTeb1oJaLy7W17dxlezadV5v0Yu0ia#scrollTo=kQn6fbhgtH9I

https://github.com/sanzmalex/AlgoritmosdeOptimizacion

Torres de Hanoi con Divide y vencerás

```
def Torres_Hanoi(N, desde, hasta):
  if N ==1 :
    print("Lleva la ficha " ,desde , " hasta ", hasta )

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


Torres_Hanoi(3, 1 , 3)
```

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

```
#Sucesión_de_Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la suscesión de Fibonacci
def Fibonacci(N:int):
  if N < 2:
    return 1
  else:
    return Fibonacci(N-1)+Fibonacci(N-2)

Fibonacci(5)
```

```
8
```

Devolución de cambio por técnica voraz

```
def cambio_monedas(N, SM):
  SOLUCION = [0]*len(SM)   #SOLUCION = [0,0,0,0,..]
  ValorAcumulado = 0

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

    if ValorAcumulado == N:
      return SOLUCION


cambio_monedas(15,[25,10,5,1])
```

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

N-Reinas por técnica de vueta atrás

```
def escribe(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="")


def es_prometedora(SOLUCION,etapa):
  #print(SOLUCION)
  #Si la solución tiene dos valores iguales no es valida => Dos reinas en la mis
  for i in range(etapa+1):
    #print("El valor " + str(SOLUCION[i]) + " está " +  str(SOLUCION.count(SOLUC
    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



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)
            #escribe(solucion)
            print()
          else:
            reinas(N, solucion, etapa+1)
        else:
          None

        solucion[etapa] = 0

    reinas(8)
```

```
[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]
```

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]
]



######################################################################
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 po
        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 ""
  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:
',0,2,5'
```

Fibonacci O(n)

La función que se ha usado antes para calcular el término n-ésimo de la secuencia de Fibonacci tiene orden de complejidad exponencial, veamos una forma de que tenga orden lineal, es decir, O(n):

```
def Fibonacci_lineal(n):
  if n<2: #Si n<2 estamos en uno de los dos primero elementos de la secuencia
    return 1
  f0, f1 = 1, 1
  for i in range(2, n+1):
    f0, f1=f1, f0+f1 #Actualizamos a la vez f0 para que tome el valor de f1, y f
  return f1
```

```
print(Fibonacci_lineal(1))
print(Fibonacci_lineal(58))
```

```
1
956722026041
```

Los dos puntos más cercanos

```
#Creamos tres listas de 10000 puntos aleatorios de dimensiones 1D, 2D y 3D
import random

LISTA_1D = [random.randrange(1,10000) for x in range(1000)]
```

Primero vamos a resolver el problema de encontrar los dos puntos más cercanos usando la fuerza bruta:

```
def Dos_Puntos_1D_FB(lista):
  n=len(lista)
  if n < 2: #Si la lista tiene solo un punto no tiene sentido continuar
    return None

  punto1=0 #Inicializamos los puntos
  punto2=0
  distancia=float('Inf') #Distancia incial infinita
  for i in range(n): #Iteramos sobre el índice de la lista
    for j in range(i+1,n): #Iteramos sobre el resto de índices, descartando los
      distancia2 = abs(lista[i]-lista[j]) #Calculamos distancia
      if distancia2 < distancia: #Si hemos mejorado la distancia, la guardamos j
        distancia = distancia2
        punto1=lista[i]
        punto2=lista[j]
  print(f"Los puntos más cercanos son {punto1} y {punto2}.")
  return punto1, punto2

print(Dos_Puntos_1D_FB([3403, 4537, 9089, 9746, 7259]))
print(Dos_Puntos_1D_FB(LISTA_1D))
```

```
Los puntos más cercanos son 9089 y 9746.
(9089, 9746)
Los puntos más cercanos son 7733 y 7733.
(7733, 7733)
```

El orden de complejidad de la función anterior es $O(n^2)$, siendo n la longitud de la lista, ya que encontramos un bucle dentro de otro y dado la cantidad restante de operaciones que se realizan dentro de la función es despreciable para n lo suficientemente grande, es decir, porque es una cantidad constante de operaciones.

A continuación resolvemos el problema usando Divide y Vencerás:

```
def Dos_Puntos_1D_DV(lista):
  if len(lista)<2: #Si la lista tiene solo un punto no tiene sentido continuar
    return None
  else:
    lista_ordenada=sorted(lista) #Complejidad entre O(n) y O(nlogn)
    p1, p2, d = Dos_Puntos_1D_DV_ordenanda(lista_ordenada)
    print(f"Los puntos más cercanos son {p1} y {p2}.")
    return p1, p2

def Dos_Puntos_1D_DV_ordenanda(lista):
  n=len(lista)
  if n < 2:
    return None, None, None
  if n==2:
    return lista[0], lista[1], abs(lista[0]-lista[1])

  medio = n//2 #Calculamos el punto medio, redondeando hacia abajo si es impar
  punto1=lista[medio-1]
  punto2=lista[medio]
  distancia=abs(punto1-punto2)

  p_primera_1, p_primera_2, distacia_primera_mitad=Dos_Puntos_1D_DV_ordenanda(li
  p_segunda_1, p_segunda_2, distacia_segunda_mitad=Dos_Puntos_1D_DV_ordenanda(li
  #El orden del slicing es O(n)

  if distacia_primera_mitad != None and distacia_primera_mitad<distancia:#Si la
    distancia=distacia_primera_mitad
    punto1=p_primera_1
    punto2=p_primera_2
  if distacia_segunda_mitad != None and distacia_segunda_mitad<distancia:#Si la
    distancia=distacia_segunda_mitad
    punto1=p_segunda_1
    punto2=p_segunda_2
  return punto1, punto2, distancia

print(Dos_Puntos_1D_DV([3403, 4537, 9089, 9746, 7259]))
print(Dos_Puntos_1D_FB(LISTA_1D))
```

```
Los puntos más cercanos son 9089 y 9746.
(9089, 9746)
Los puntos más cercanos son 7733 y 7733.
(7733, 7733)
```

El orden de complejidad de la función anterior es $O(n\ log(n))$, siendo n la longitud de la lista, ya que tanto la función sorted como el slicing (repetido $log(n)$ veces debido al método de divide y vencerás) tienen orden $O(n\ log(n))$, siendo el resto de operaciones de orden constante (aunque repetidas debido debido al método de divide y vencerás).

Referencias

-Complejidad sorted()
-Complejidad slicing