AG - Actividad guiada 1

Víctor Cebrián

URL Git: https://github.com/vcebrian/03MAIR---Algoritmos-de-optimizacion/tree/master/AG2

```
#Decorador para calcular tiempos de ejecución
from time import time

#Función para calcular el tiempo de ejecución
def calcular_tiempo(f):
    def wrapper(*args, **kwargs):
        inicio = time()
        resultado = f(*args, **kwargs)
        tiempo = time() - inicio
        print("Tiempo de ejecución para algoritmo: "+str(tiempo))
        return resultado

    return wrapper


import math
import random

def distancia(A,B):
  if type(A) is int or type(A) is float:
    return abs(A-B)
  else:
    return math.sqrt(sum( [(A[i]-B[i])**2 for i in range(len(A))] ))



N=3000

LISTA_2D = [(random.randrange(1,N*10),random.randrange(1,N*10)) for _ in range(N)]


#Puntos más cercanos por fuerza bruta
def distancia_fuerza_bruta(A):
  solucion = [A[0],A[1]]
  distancia_a_mejorar = distancia(solucion[0],solucion[1])
  for i in range(len(A)):
    for j in range(i+1, len(A)):
        if distancia(A[i],A[j])<distancia_a_mejorar:
          solucion = [A[i],A[j]]
          distancia_a_mejorar = distancia(solucion[0],solucion[1])

  return solucion

distancia_fuerza_bruta(LISTA_2D)
```

```
[(11014, 7008), (11022, 7010)]
```

```
@calcular_tiempo
def lanza(L):
  print(distancia_fuerza_bruta(L))

lanza(LISTA_2D)
```

```
[(11014, 7008), (11022, 7010)]
Tiempo de ejecución para algoritmo: 9.655901670455933
```

```
def distancia_divide_y_venceras(A):
  #con pocos elementos, aplico fuerza bruta
  if len(A)<10:
```

```
    return distancia_fuerza_bruta(A)

  #Dividimos en listas grandes

  LISTA_IZQ = sorted(A, key=lambda x: x[0])[:len(A)//2]
  LISTA_DER = sorted(A, key=lambda x: x[0])[len(A)//2:]

  PUNTOS_LISTA_IZQ = distancia_divide_y_venceras(LISTA_IZQ)
  PUNTOS_LISTA_DER = distancia_divide_y_venceras(LISTA_DER)

  return distancia_divide_y_venceras(PUNTOS_LISTA_IZQ+PUNTOS_LISTA_DER)



@calcular_tiempo
def lanza(L):
  print(distancia_divide_y_venceras(L))

lanza(LISTA_2D)
```

```
[(11014, 7008), (11022, 7010)]
Tiempo de ejecución para algoritmo: 0.05153346061706543
```

```
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]
]

#Paseo por el rio

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(N-1):
    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(PRECIOS[i][k]+TARIFAS[k][j], MIN)
          RUTA[i][j] = k

      PRECIOS[i][j] = MIN

  return PRECIOS, RUTA


PRECIOS, RUTAS = Precios(TARIFAS)

print(PRECIOS)
print()
print(RUTAS)
```

```
[[9999, 5, 4, 3, 8, 8, 11], [9999, 9999, 999, 2, 3, 8, 7], [9999, 9999, 9999,

[['', 0, 0, 0, 1, 2, 5], ['', '', 1, 1, 1, 3, 4], ['', '', '', 2, 3, 2, 5], |
```

```
def calcular_ruta(RUTAS, desde, hasta):
  if desde == hasta:
```

```
      return desde
    else:
      return str(calcular_ruta(RUTAS,desde,RUTAS[desde][hasta])) + ',' + str (RUTAS[d

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

```
La ruta es:
'0,0,2,5'
```