

Nombre: Salvador Gimeno

03MAIR_Algoritmos de Optimizacion - Actividad Guiada 1

Google Drive: <https://colab.research.google.com/drive/1zYq-b8JvamnReuLOAJqr6cs7jPscJLO?usp=sharing>

Github: <https://github.com/salvagimeno-ai/03MAIR-Algoritmos-de-optimizacion/tree/master/AG1>

1) Decorador para medir tiempos

```
from functools import wraps
from time import time

def calcular_tiempo(f):
    @wraps(f)
    def cronometro(*args, **kwargs):
        t_inicial = time()
        salida = f(*args, **kwargs)
        t_final = time()
        print('Tiempo transcurrido (en segundos): {}'.format(t_final - t_inicial))
        return salida
    return cronometro
```

2) Problema: Torres de Hanoy - Técnica: Divide y vencerás

```
def torres_hanoy(n, desde, hasta):
    if n == 1:
        print('llevar desde ' + str(desde) + ' hasta ' + str(hasta))
```

```

else:
    torres_hanoy2(n-1,desde,6-desde-hasta)
    print('llevar desde ' + str(desde) + ' hasta ' + str(hasta))
    torres_hanoy2(n-1,6-desde-hasta,hasta)

```

```
torres_hanoy(3,1,3)
```

```

↳ llevar desde 1 hasta 3
Mueve el disco 2 desde 1 hasta 2
llevar desde 3 hasta 2
llevar desde 1 hasta 3
llevar desde 2 hasta 1
Mueve el disco 2 desde 2 hasta 3
llevar desde 1 hasta 3

```

Otra opción para resolver el problema de las Torres de Hanoy

```

def torres_hanoy2(n, desde=1, hasta=3):
    if n:
        torres_hanoy(n-1, desde, 6-desde-hasta)
        print("Mueve el disco {} desde {} hasta {}".format(n, desde, hasta))
        torres_hanoy(n-1, 6-desde-hasta, hasta)

```

```
torres_hanoy2(3)
```

```

↳ Mueve el disco 1 desde 1 hasta 3
Mueve el disco 2 desde 1 hasta 2
Mueve el disco 1 desde 3 hasta 2
Mueve el disco 3 desde 1 hasta 3
Mueve el disco 1 desde 2 hasta 1
Mueve el disco 2 desde 2 hasta 3
Mueve el disco 1 desde 1 hasta 3

```

3) Ordenacion con Algoritmo de Quick Sort con técnica Divide y Vencerás

```
import random
def quick_sort(A):
    if len(A) == 1:
        return A

    elif len(A) == 2:
        return [min(A),max(A)]

    elif len(A) > 2:
        #en este caso el pivote se calculará como la media de los 3 primeros valores de la lista
        pivote = (A[0] + A[1] + A[2])/3

        IZQ = []
        DER = []

        for i in A:
            if i < pivote:
                IZQ.append(i)
            else:
                DER.append(i)
        #print('paso recursivo')
        return quick_sort(IZQ) + quick_sort(DER)

# LISTAS PARA PRUEBAS:
A = [9187, 244, 4054, 9222, 8373, 4993, 5265, 5470, 4519, 7182, 2035, 3506, 4337, 7580, 2554, 2824, 8357, 4447, 7379]
B = [9187, 244, 1, 24, 154, 2321, 123, 12]
C = [9187]


@calcular_tiempo
def ordenar(A):
    print(quick_sort(A))

ordenar(A)
```



```
[244, 2035, 2554, 2824, 3506, 4054, 4337, 4447, 4519, 4993, 5265, 5470, 7182, 7379, 7580, 8357, 8373, 9187, 9222]
```

```
D=list(map(lambda x: random.randrange(1, 10000), range(1,300)))
ordenar(D)
```

 [56, 79, 110, 114, 126, 133, 147, 241, 246, 258, 293, 295, 311, 345, 452, 484, 499, 536, 577, 635, 643, 673, 675, 715
 Tiempo transcurrido (en segundos): 0.0010120868682861328

4) Problema: Cambio de monedas - Técnica: Algoritmo Voraz

```
Sistema_Monetario=[25,10,5,1]

@calcular_tiempo
def cambio_monedas1(N,Sistema_Monetario):
    ValorAcumulado = 0
    #SOLUCION = [0 for i in range(len(Sistema_Monetario))] # inicializamos los elementos del array solucion
    SOLUCION = [0]*len(Sistema_Monetario)

    for i,m in enumerate(Sistema_Monetario):
# for i in range(len(Sistema_Monetario)):
        monedas = (N - ValorAcumulado)//m
        ValorAcumulado = ValorAcumulado + monedas*m
        SOLUCION[i] = monedas

    if ValorAcumulado == N:
        return SOLUCION

    return SOLUCION

cambio_monedas1(77,Sistema_Monetario)

[➤] Tiempo transcurrido (en segundos): 4.0531158447265625e-06
[3, 0, 0, 2]
```

Otra opción para resolver el problema de Cambio de Monedas

```
Sistema_Monetario=[25,10,5,1]

@calcular_tiempo
def cambio_monedas2(N,Sistema_Monetario):
    SOLUCION = [0 for i in range(len(Sistema_Monetario))] # inicializamos los elementos del array solucion

    ValorAcumulado = 0

    for i in range(len(Sistema_Monetario)):
        monedas = int((N - ValorAcumulado)/Sistema_Monetario[i])

        SOLUCION[i] = monedas
        ValorAcumulado += monedas*Sistema_Monetario[i]

        if N == ValorAcumulado:
            return SOLUCION

cambio_monedas2(77,Sistema_Monetario)

↳ Tiempo transcurrido (en segundos): 8.58306884765625e-06
[3, 0, 0, 2]
```

Resumen de técnicas utilizadas:

```
# 1) Inicializacion de los elementos del array SOLUCION
Sistema_Monetario=[25,10,5,1]
SOLUCION = [0 for i in range(len(Sistema_Monetario))] # inicializamos los elementos del array solucion
print(SOLUCION)

↳ [0, 0, 0, 0]
```

```
# 2) Funcion ENUMERATE: nos permite iterar sobre un array, y nos genera una variable que nos sirve de índice de cada uno
# de los valores del array
for i,m in enumerate(Sistema_Monetario):
    print(i,m)
```

☞

```
0 25
1 10
2 5
3 1
```

Problema de las 4 Reinas - Técnica: Vuelta atrás (Backtracking)

N=4

Solucion0=[0 for i in range(N)] # inicializamos el array

Etapas=0

```
def es_prometedora(Solucion,Etapas):
    #print(Solucion)
    for i in range (Etapas+1):
        if Solucion.count(Solucion[i])>1:
            return False
        for j in range(i+1,Etapas+1):
            if abs(i-j)== abs(Solucion[i]-Solucion[j]): return False
    return True
```

```
def Dibuja(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 Reinas (N,Solucion=Solucion0,Etapa=0):
    for i in range(1,N+1):
        Solucion[Etapa]=i
        EsPrometedora=es_prometedora(Solucion,Etapa)

        if EsPrometedora and Etapa==N-1:
            print ("\n\nla solución es:")
            print (Solucion)
            Dibuja(Solucion)
        elif EsPrometedora:
            Reinas(N,Solucion,Etapa+1)
        else:
            None
            Solucion[Etapa]=0

@calcular_tiempo
def TR(N):
    return Reinas(N)
```

TR(N)



Nota: Como generar conjuntos de datos aleatorios:

```
import random
n = 10
LISTA_1D = [random.randrange(1,n) for i in range(n)]
print(LISTA_1D)
```

```
↳ [7, 3, 4, 9, 5, 1, 3, 2, 4, 1]
    - x - -
```

```
import random
n = 10
LISTA_2D = [(random.randrange(1,n), random.randrange(1,n)) for i in range(n)]
print(LISTA_2D)
```

```
↳ [(3, 1), (5, 6), (8, 7), (4, 7), (8, 6), (6, 9), (2, 1), (1, 9), (5, 6), (2, 2)]
```

Práctica Individual: Encontrar los dos puntos más cercanos

Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos. Primer intento Fuerza Bruta. Calcular la complejidad. Segundo intento: Divide y vencerás. Calcular la complejidad.

```
import random
n = 20
P = [random.randrange(1,n) for i in range(n)]
print(P)
```

```
↳ [11, 4, 4, 12, 19, 12, 15, 5, 17, 7, 18, 6, 18, 14, 11, 14, 17, 8, 8, 4]
```

```
# calculo distancia entre dos puntos
import numpy as np
```



```
def dist_e(x, y):
    return np.sqrt(np.sum((x - y) ** 2))

# ejemplo: dist_e(2,5)
```

a) Fuerza Bruta:

- En este caso lo que haremos será "**calcular las distancias entre todos los pares de puntos del conjunto y seleccionar el mínimo**".
- Requiere un tiempo **$O(n^2)$** .

```
import math
minDist = math.inf
for i in P:
    for j in P:
        if i != j and dist_e(i,j)< minDist:
            minDist = dist_e(i,j)
            ClosestPair = (i, j)

print(ClosestPair)
```

```
↳ (11, 12)
```

b) Divide y Vencerás:

1. Ordenar los puntos según su coordenada X.
2. Si el tamaño del conjunto es 2, devolver la distancia entre ellos. Si el conjunto tiene 0 o 1 elementos, devolver infinito.
3. Dividir el conjunto de puntos en dos partes iguales (del mismo número de puntos).
4. Solucionar el problema de forma recursiva en las partes izquierdas y derecha. Esto devolverá una solución para cada parte, llamadas dLmin y dRmin. Escoger el mínimo entre estas dos soluciones, llamado dLRmin.

5. Seleccionar los puntos de la parte derecha e izquierda que están a una distancia horizontal menor que dLR_{min} de la recta divisoria entre ambos.
6. Aprovechar que los puntos están ordenados para elegir los últimos puntos de la parte izquierda y los primeros de la parte derecha.
7. Encontrar la distancia mínima dC_{min} entre todos los pares de puntos formados por un punto de cada parte del paso anterior.
8. La respuesta final es el mínimo entre dC_{min} y dLR_{min}

#

Tiempo de Ejecución:

$T(n)$ = tiempo de ejecución de cada etapa recursiva $T'(n)$ = tiempo de ejecución del algoritmo total

$$T'(n) = T(n) + O(n \lg n)$$

$$T(n) = 2T(n/2) + O(n), \text{ si } n > 3 \quad T(n) = O(1), \text{ si } n \leq 3$$

Entonces, $T(n) = O(n \lg n)$, y $T'(n) = O(n \lg n)$

Conclusión: **el algoritmo Divide y Vencerás $O(n \lg n)$ es mucho más eficiente que el algoritmo de Fuerza Brute $O(n^2)$.**

Referencia: https://es.qwe.wiki/wiki/Divide-and-conquer_algorithm

