Algoritmos - Luis Jama - AG1

July 19, 2023

1 Actividad Guiada 1

https://github.com/ljham/03MIAR_ALG_OPTZ.git

2 Luis Jama Tello

1. Desarrollo de algoritmo con la técnica de divide y vencerás (Torres de Hanoi)

[ ]:
```
def torres_hanoi(n, origen, destino, pivote):
    if(n == 1):
        print(f'Mover bloque desde {origen} a {destino} ')
        return

    torres_hanoi(n-1, origen=origen, destino=pivote, pivote=destino)
    print(f"Mover bloque desde {origen} a {destino}")
    torres_hanoi(n-1, origen=pivote, destino=destino, pivote=origen)

torres_hanoi(4, 1, 3, 2)
```

```
Mover bloque desde 1 a 2
Mover bloque desde 1 a 3
Mover bloque desde 2 a 3
Mover bloque desde 1 a 2
Mover bloque desde 3 a 1
Mover bloque desde 3 a 2
Mover bloque desde 1 a 2
Mover bloque desde 1 a 3
Mover bloque desde 2 a 3
Mover bloque desde 2 a 1
Mover bloque desde 3 a 1
Mover bloque desde 2 a 3
Mover bloque desde 1 a 2
Mover bloque desde 1 a 3
Mover bloque desde 2 a 3
```

2. Desarrollo de algoritmo voraz para resolver problemas (Devolución de cambio)

[ ]:
```python
def cambio_moneda(cantidad, sistema):
    print('Sistema : ')
    print(sistema)

    # Crea una lista con ceros de acuerdo a la dimension del sistema
    solucion = [0 for i in range(len(sistema))]
    valor_acumulado = 0

    for i in range(len(sistema)):
        monedas = int((cantidad - valor_acumulado)/sistema[i])
        solucion[i] = monedas
        valor_acumulado += monedas * sistema[i]
        if(valor_acumulado == cantidad): break

    return solucion

# Sistema de monedas
sistema = [25, 10, 5, 1]

# Cambio
print(cambio_moneda(19, sistema))
```

```
Sistema :
[25, 10, 5, 1]
[0, 1, 1, 4]
```

3. Problema : Encontrar los dos puntos más cercanos

- Primer intento : Fuerza Bruta

[ ]:
```python
def puntos_mas_cercanos_fuerza_bruta(puntos):
    n = len(puntos)
    min_distancia = float('inf')
    punto1 = punto2 = None

    for i in range(n - 1):
        for j in range(i + 1, n):
            dist = abs(puntos[j] - puntos[i])
            if dist < min_distancia:
                min_distancia = dist
                punto1 = puntos[i]
                punto2 = puntos[j]

    return punto1, punto2
```

[ ]:
```python
import random

puntos_1d = [random.randrange(1, 10000) for x in range(10)]
```

```
print(puntos_1d)

punto_cercano1, punto_cercano2 = puntos_mas_cercanos_fuerza_bruta(puntos_1d)

print("Punto 1:", punto_cercano1)
print("Punto 2:", punto_cercano2)
print("Distancia:", abs(punto_cercano2 - punto_cercano1))
```

```
[8439, 9425, 5492, 1729, 1600, 3537, 337, 4421, 4288, 487]
Punto 1: 1729
Punto 2: 1600
Distancia: 129
```

La compleijidad para este algoritmo es: O(n^2) donde n es el número de puntos.

- Segundo intento : Divide y vencerás

[ ]:
```
import math

def puntos_mas_cercanos_divide_venceras(puntos):
    puntos_ordenados = sorted(puntos)  # Ordenar los puntos en orden ascendente
    return puntos_mas_cercanos(puntos_ordenados)

def puntos_mas_cercanos(puntos):
    n = len(puntos)

    # Caso base: Si hay 2 puntos, retornar directamente
    if n == 2:
        return puntos[0], puntos[1]

    # Caso base: Si hay 3 puntos, calcular distancias y retornar los dos puntos␣
 ↪más cercanos
    if n == 3:
        dist1 = abs(puntos[1] - puntos[0])
        dist2 = abs(puntos[2] - puntos[1])

        if dist1 < dist2:
            return puntos[0], puntos[1]
        else:
            return puntos[1], puntos[2]

    # Dividir en mitades y encontrar los dos puntos más cercanos en cada mitad
    mitad = n // 2
    izquierda = puntos[:mitad]
    derecha   = puntos[mitad:]

    punto_izq1, punto_izq2 = puntos_mas_cercanos(izquierda)
    punto_der1, punto_der2 = puntos_mas_cercanos(derecha)
```

```
    # Encontrar los dos puntos más cercanos entre los resultados de cada mitad
    dist_izq = abs(punto_izq2 - punto_izq1)
    dist_der = abs(punto_der2 - punto_der1)
    min_distancia = min(dist_izq, dist_der)

    punto1 = punto_izq1
    punto2 = punto_izq2

    # Comprobar si hay un punto más cercano entre las mitades
    for i in range(mitad-1, -1, -1):
        dist = abs(puntos[i+1] - puntos[i])
        if dist < min_distancia:
            min_distancia = dist
            punto1 = puntos[i]
            punto2 = puntos[i+1]
        else:
            break

    return punto1, punto2
```

[ ]:
```
import random

puntos_1d = [random.randrange(1, 10000) for x in range(10)]
print(puntos_1d)

punto_cercano1, punto_cercano2 = puntos_mas_cercanos_divide_venceras(puntos_1d)

print("Punto 1:", punto_cercano1)
print("Punto 2:", punto_cercano2)
print("Distancia:", abs(punto_cercano2 - punto_cercano1))
```

```
[4545, 3825, 627, 4290, 2143, 7343, 4708, 4316, 5222, 9056]
Punto 1: 627
Punto 2: 2143
Distancia: 1516
```

La complejidad de este algoritmo de divide y vencerás es O(n log n), ya que se realiza una ordenación inicial de los puntos, que tiene una complejidad de O(n log n), y luego se realiza una división en mitades y llamadas recursivas para encontrar los puntos más cercanos en cada mitad, lo cual también tiene una complejidad de O(n log n).