LUIS_GUAMAN-AG1

May 25, 2024

#Primera clase guiada

ACTIVIDAD GUIADA NRO 1 ALGORITMOS DE OPTIMIZACIÓN Nombre: LUIS R. GUAMAN Link de Git Hub: https://github.com/gluishs/03MIAR_04_2024_25_Algoritmos_Optimizacion_LG/ Link GoogleColab: https://colab.research.google.com/drive/1ir9vLEh3dbXXZso6HN1x2unOm9G7jPH-?usp=sharing

DIVIDE Y VENCERAS TORRES DE HANOI

[ ]:
```
#Torres de Hanoi - Divide y venceras
###############################################################################



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

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

Torres_Hanoi(3, 1, 3)
#################################################################
#PUNTOS EXTRAS SI SE HACE ALGO MAS COMO EJEMPLO CON FOR
```

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

Algortimos voraces Problema cambio emonedas

[ ]:
```
SISTEMA = [25,10,5,1]
  #probar para distintos valores de N y probar con distintos sistemas
```

[ ]:
```
#Cambio de monedas - Técnica voraz
###############################################################################
SISTEMA = [11, 5 ,  1  ]
#####################################################################
def cambio_monedas(CANTIDAD,SISTEMA):
#....
  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:
      return SOLUCION

  print("No es posible encontrar solucion")

#cambio_monedas(15,SISTEMA)
print(cambio_monedas(15,[11, 5 ,  1  ]))#Para 15$ devuelve en la pos 1 value 11
 ↪+ pos 4 value 4
cambio_monedas(2,[11, 5 ,  1  ])#Para $2 devuelve el la posion(2) 2 veces


#####################################################################
```

```
[1, 0, 4]
```

[ ]:
```
[0, 0, 2]
```

Algoritmo de Técnica Vuelta Atras:Bactracking

[ ]:
```
#N Reinas - Vuelta Atrás()
###############################################################################

#Verifica que en la solución parcial no hay amenzas entre reinas
#####################################################################
def es_prometedora(SOLUCION,etapa):
#####################################################################
  #print(SOLUCION, etapa)
  #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))
      #print(abs(i-j), abs(SOLUCION[i]-SOLUCION[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="")



#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(4,solucion=[],etapa=0)
#reinas(8)
```

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

 -  -  X  -
 X  -  -  -
 -  -  -  X
 -  X  -  - [3, 1, 4, 2]

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

PROGRAMACION DINAMICA Problema: Viaje por el rio

[ ]:
```
#Viaje por el rio - Programacion dinamica
def Precios(TARIFAS):
  #Total de Nodos
  N = len(TARIFAS[0])

  #Inicializacion de la tabla de precios
  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(MIN, PRECIOS[i][k] + TARIFAS[k][j])
          RUTA[i][j] = k
        PRECIOS[i][j] = MIN
  return PRECIOS, RUTA

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]
]
```

```
PRECIOS = Precios(TARIFAS)

#RUTA = Precios(TARIFAS)

def calcular_ruta(RUTA, desde, hasta):
  #print(RUTA)
  if desde == hasta:
    #print("Ir a :" + str(desde))
    return desde
  else:
    return str(calcular_ruta(RUTA, desde, RUTA[desde][hasta])) + ',' +␣
 ↪str(RUTA[desde][hasta])


print("\n La RUTA es:")
print(calcular_ruta(PRECIOS[1], 0,6))

PRECIOS
```

```
 La RUTA es:
0,0,2,5
```

```
[ ]: ([[9999, 5, 4, 3, 8, 8, 11],
   [9999, 9999, 999, 2, 3, 8, 7],
   [9999, 9999, 9999, 1, 6, 4, 7],
   [9999, 9999, 9999, 9999, 5, 6, 9],
   [9999, 9999, 9999, 9999, 9999, 999, 4],
   [9999, 9999, 9999, 9999, 9999, 9999, 3],
   [9999, 9999, 9999, 9999, 9999, 9999, 9999]],
  [['', 0, 0, 0, 1, 2, 5],
   ['', '', 1, 1, 1, 3, 4],
   ['', '', '', 2, 3, 2, 5],
   ['', '', '', '', 3, 3, 3],
   ['', '', '', '', '', 4, 4],
   ['', '', '', '', '', '', 5],
   ['', '', '', '', '', '', '']])
```

PRÁCTICA INDIVIDUAL Problema: Encontrar los dos puntos más cercanos • Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos • Guía para aprendizaje: Suponer en 1D, o sea, una lista de números: [3403, 4537, 9089, 9746, 7259, … Primer intento: Fuerza bruta Calcular la complejidad. ¿Se puede mejorar? Segundo intento. Aplicar Divide y Vencerás Calcular la complejidad. ¿Se puede mejorar? Extender el algoritmo a 2D: [(1122, 6175), (135, 4076), (7296, 2741)… Extender el algoritmo a 3D.

Fuerza Bruta La función de fuerza bruta compara cada par de puntos y calcula la distancia entre ellos. Para n puntos, la complejidad temporal es: (2) O(n 2) Divide y Vencerás La función de

"Divide y Vencerás" es mucho más eficiente para grandes conjuntos de datos. Este enfoque sigue los siguientes pasos:

Divide los puntos en dos mitades. Encuentra los puntos más cercanos en cada mitad recursivamente. Combina los resultados para encontrar los puntos más cercanos en el conjunto original. La complejidad temporal de este algoritmo es:

(log) O(nlogn)

ALGORITMOS POR FUERZA BRUTA PARA 1D, 2D Y 3D

[ ]:
```python
#1D
#Problema: Encontrar los dos puntos más cercanos
#FUERZA BRUTA
def puntos_cercanos_fuerza_bruta_1d(points):
    distancia_minima = max(points) #Distancia mínima entre dos puntos
    p1, p2 = None, None #Puntos cercanos
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            distancia = abs(points[i] - points[j])
            if distancia < distancia_minima:
                distancia_minima = distancia
                p1, p2 = points[i], points[j]
    return p1, p2, distancia_minima
```

[ ]:
```python
#2D
#Problema: Encontrar los dos puntos más cercanos
#FUERZA BRUTA
def distancia_2d(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def puntos_cercanos_fuerza_bruta_2d(points):
    distancia_minima = float('inf') #Distancia mínima entre dos puntos
    p1, p2 = None, None #Puntos cercanos
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            distancia = distancia_2d(points[i], points[j])
            if distancia < distancia_minima:
                distancia_minima = distancia
                p1, p2 = points[i], points[j]
    return p1, p2, distancia_minima
```

[ ]:
```python
#3D
#Problema: Encontrar los dos puntos más cercanos
#FUERZA BRUTA
import math
def distancia_3d(p1, p2):
```

```
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2] -␣
 ↪p2[2])**2)

def puntos_cercanos_fuerza_bruta_3d(points):
    distancia_minima = float('inf') #Distancia mínima entre dos puntos
    p1, p2 = None, None #Puntos cercanos
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            distancia = distancia_3d(points[i], points[j])
            if distancia < distancia_minima:
                distancia_minima = distancia
                p1, p2 = points[i], points[j]
    return p1, p2, distancia_minima
```

ALGORITMOS DIVIDE Y VENCERAS PARA 1D, 2D Y 3D

[ ]:
```
#1D
#Problema: Encontrar los dos puntos más cercanos
#DIVIDE Y VENCERAS
def puntos_cercanos_divide_venceras_1d(points):
    def par_cercano_rec_1d(puntos_ordenados):
        if len(puntos_ordenados) <= 3:
            return puntos_cercanos_fuerza_bruta_1d(puntos_ordenados)
        mitad = len(puntos_ordenados) // 2
        valor_punto_medio = puntos_ordenados[mitad]
        left = puntos_ordenados[:mitad]
        right = puntos_ordenados[mitad:]

        p1, p2, dist1 = par_cercano_rec_1d(left)
        q1, q2, dist2 = par_cercano_rec_1d(right)

        min_dist = min(dist1, dist2)
        if dist1 < dist2:
            mejor_par = (p1, p2)
        else:
            mejor_par = (q1, q2)

        strip = [p for p in puntos_ordenados if abs(p - valor_punto_medio) <␣
 ↪min_dist]

        for i in range(len(strip)):
            for j in range(i + 1, min(i + 7, len(strip))):
                dist = abs(strip[i] - strip[j])
                if dist < min_dist:
                    min_dist = dist
                    mejor_par = (strip[i], strip[j])
```

```
        return mejor_par[0], mejor_par[1], min_dist

    puntos_ordenados = sorted(points)
    return par_cercano_rec_1d(puntos_ordenados)
```

[ ]:
```
#2D
#Problema: Encontrar los dos puntos más cercanos
#DIVIDE Y VENCERAS

def puntos_cercanos_divide_venceras_2d(points):
    def par_cercano_rec_2d(sorted_x, sorted_y):
        if len(sorted_x) <= 3:
            return puntos_cercanos_fuerza_bruta_2d(sorted_x)

        mitad = len(sorted_x) // 2
        valor_punto_medio = sorted_x[mitad]

        sorted_y_left = [p for p in sorted_y if p[0] <= valor_punto_medio[0]]
        sorted_y_right = [p for p in sorted_y if p[0] > valor_punto_medio[0]]

        p1, q1, dist1 = par_cercano_rec_2d(sorted_x[:mitad], sorted_y_left)
        p2, q2, dist2 = par_cercano_rec_2d(sorted_x[mitad:], sorted_y_right)

        if dist1 < dist2:
            d = dist1
            mejor_par = (p1, q1)
        else:
            d = dist2
            mejor_par = (p2, q2)

        strip = [p for p in sorted_y if abs(p[0] - valor_punto_medio[0]) < d]

        for i in range(len(strip)):
            for j in range(i + 1, min(i + 7, len(strip))):
                dist = distancia_2d(strip[i], strip[j])
                if dist < d:
                    d = dist
                    mejor_par = (strip[i], strip[j])

        return mejor_par[0], mejor_par[1], d

    sorted_x = sorted(points, key=lambda x: x[0])
    sorted_y = sorted(points, key=lambda x: x[1])
    return par_cercano_rec_2d(sorted_x, sorted_y)
```

[ ]:
```python
#3D
#Problema: Encontrar los dos puntos más cercanos
#DIVIDE Y VENCERAS

def puntos_cercanos_divide_venceras_3d(points):
    def par_cercano_rec_3d(sorted_x, sorted_y, sorted_z):
        if len(sorted_x) <= 3:
            return puntos_cercanos_fuerza_bruta_3d(sorted_x)

        mitad = len(sorted_x) // 2
        valor_punto_medio = sorted_x[mitad]

        sorted_y_left = [p for p in sorted_y if p[0] <= valor_punto_medio[0]]
        sorted_y_right = [p for p in sorted_y if p[0] > valor_punto_medio[0]]

        sorted_z_left = [p for p in sorted_z if p[0] <= valor_punto_medio[0]]
        sorted_z_right = [p for p in sorted_z if p[0] > valor_punto_medio[0]]

        p1, q1, dist1 = par_cercano_rec_3d(sorted_x[:mitad], sorted_y_left,␣
 ↪sorted_z_left)
        p2, q2, dist2 = par_cercano_rec_3d(sorted_x[mitad:], sorted_y_right,␣
 ↪sorted_z_right)

        if dist1 < dist2:
            d = dist1
            mejor_par = (p1, q1)
        else:
            d = dist2
            mejor_par = (p2, q2)

        strip = [p for p in sorted_y if abs(p[0] - valor_punto_medio[0]) < d]

        for i in range(len(strip)):
            for j in range(i + 1, min(i + 7, len(strip))):
                dist = distancia_3d(strip[i], strip[j])
                if dist < d:
                    d = dist
                    mejor_par = (strip[i], strip[j])

        return mejor_par[0], mejor_par[1], d

    sorted_x = sorted(points, key=lambda x: x[0])
    sorted_y = sorted(points, key=lambda x: x[1])
    sorted_z = sorted(points, key=lambda x: x[2])
    return par_cercano_rec_3d(sorted_x, sorted_y, sorted_z)
```

**

Generación de Puntos y Prueba de los Algoritmos

**

[ ]:
```
#Se generan randomicamente los puntso que seran evaluados con los algortimos
 ↪propuestos
import random

# Generar puntos únicos en 1D
LISTA_1D = list(set(random.randrange(1, 10000) for _ in range(1000)))

# Generar puntos únicos en 2D
LISTA_2D = list(set((random.randrange(1, 10000), random.randrange(1, 10000))
 ↪for _ in range(1000)))

# Generar puntos únicos en 3D
LISTA_3D = list(set((random.randrange(1, 10000), random.randrange(1, 10000),
 ↪random.randrange(1, 10000)) for _ in range(1000)))
```

PRUEBA DE LA EFECTIVIDAD DE LOS ALGORITMOS(VERIFICACIÓN DE LA COMPLEJIDAD Y COSTES)

[ ]:
```
#Prueba de los Algoritmos
#Las pruebas se realizan con 1000 registros con un intervalo de números
 ↪randómicos entre  1 y 10000
# 1D
print("PRUEBA Fuerza Bruta - 1D ")
%time resultado_fuerza_bruta_1d = puntos_cercanos_fuerza_bruta_1d(LISTA_1D)
print(resultado_fuerza_bruta_1d)
print("PRUEBA Divide y Vencerás - 1D")
%time resultado_divide_venceras_1d =
 ↪puntos_cercanos_divide_venceras_1d(LISTA_1D)
print(resultado_divide_venceras_1d)
print("\n")

# 2D
print("PRUEBA Fuerza Bruta - 2D")
%time resultado_brute_force_2d = puntos_cercanos_fuerza_bruta_2d(LISTA_2D)
print(resultado_brute_force_2d)
print("PRUEBA Divide y Vencerás - 2D")
%time resultado_divide_and_conquer_2d =
 ↪puntos_cercanos_divide_venceras_2d(LISTA_2D)
print(resultado_divide_and_conquer_2d)
print("\n")

# 3D
print("PRUEBA Fuerza Bruta - 3D")
%time resultado_brute_force_3d = puntos_cercanos_fuerza_bruta_3d(LISTA_3D)
```

```
print(resultado_brute_force_3d)
print("PRUEBA Divide y Vencerás - 3D")
%time resultado_divide_and_conquer_3d = puntos_cercanos_divide_venceras_3d(LISTA_3D)
print(resultado_divide_and_conquer_3d)
```

```
PRUEBA Fuerza Bruta - 1D
CPU times: user 82.8 ms, sys: 0 ns, total: 82.8 ms
Wall time: 83.8 ms
(53, 54, 1)
PRUEBA Divide y Vencerás - 1D
CPU times: user 2.26 ms, sys: 0 ns, total: 2.26 ms
Wall time: 2.26 ms
(9847, 9848, 1)


PRUEBA Fuerza Bruta - 2D
CPU times: user 506 ms, sys: 0 ns, total: 506 ms
Wall time: 506 ms
((9710, 9410), (9710, 9405), 5.0)
PRUEBA Divide y Vencerás - 2D
CPU times: user 17.3 ms, sys: 0 ns, total: 17.3 ms
Wall time: 17.2 ms
((9710, 9405), (9710, 9410), 5.0)


PRUEBA Fuerza Bruta - 3D
CPU times: user 689 ms, sys: 263 µs, total: 689 ms
Wall time: 690 ms
((7332, 688, 8604), (7367, 734, 8600), 57.9396237474839)
PRUEBA Divide y Vencerás - 3D
CPU times: user 36.5 ms, sys: 0 ns, total: 36.5 ms
Wall time: 36.5 ms
((7332, 688, 8604), (7367, 734, 8600), 57.9396237474839)
```

Conclusión: La implementación del algoritmo fuerza bruta es más simple pero es menos eficiente para grandes conjuntos de datos debido a su complejidad (2) O(n 2). El algoritmo Divide y Vencerás, su enfoque es más eficiente para grandes conjuntos de datos con una complejidad (log) O(nlogn). Aunque el enfoque “Divide y Vencerás” es eficiente, se pueden considerar optimizaciones adicionales mediante el uso de estructuras de datos más avanzadas, paralelización y heurísticas específicas del problema En costes se puede observar que el algortimo divide y vencrás es mucho mas efectivo en 1D, 2D y 3D, esto debido a que por fuerza bruta no es capaz de procesar grandes catidades de conjuntos de datos.