Actividad Guiada 1 de Algoritmos de Optimizacion

Nombre: Daniel Paniagua Ares

https://github.com/dpaniaguaa/algoritmos_optimizacion_MIAR_2024

Ejercicio 1 Torres de Hanoi

In [ ]:
```
def torres_hanoi(N, desde, hasta):
  '''
  Función que simula el funcionamiento de las torres de Hanoi
  Parameters:
    N: Nº de fichas
    desde: Torre inicial
    hasta: Torre final

  '''
  if N==1 :
    print(f"Mueve la ficha {N} desde la torre nº {str(desde)} hasta la torre nº {str(hasta)}")

  else:
    torres_hanoi(N-1, desde, 6-desde-hasta)
    print(f"Mueve la ficha {N} desde la torre nº {str(desde)} hasta la torre nº {str(hasta)}")
    torres_hanoi(N-1, 6-desde-hasta,  hasta)

torres_hanoi(4, 1, 3)
```

```
Mueve la ficha 1 desde la torre nº 1 hasta la torre nº 2
Mueve la ficha 2 desde la torre nº 1 hasta la torre nº 3
Mueve la ficha 1 desde la torre nº 2 hasta la torre nº 3
Mueve la ficha 3 desde la torre nº 1 hasta la torre nº 2
Mueve la ficha 1 desde la torre nº 3 hasta la torre nº 1
Mueve la ficha 2 desde la torre nº 3 hasta la torre nº 2
Mueve la ficha 1 desde la torre nº 1 hasta la torre nº 2
Mueve la ficha 4 desde la torre nº 1 hasta la torre nº 3
Mueve la ficha 1 desde la torre nº 2 hasta la torre nº 3
Mueve la ficha 2 desde la torre nº 2 hasta la torre nº 1
Mueve la ficha 1 desde la torre nº 3 hasta la torre nº 1
Mueve la ficha 3 desde la torre nº 2 hasta la torre nº 3
Mueve la ficha 1 desde la torre nº 1 hasta la torre nº 2
Mueve la ficha 2 desde la torre nº 1 hasta la torre nº 3
Mueve la ficha 1 desde la torre nº 2 hasta la torre nº 3
```

Ejercicio 2 Cambio de monedas

In [ ]:
```
SISTEMA_ERROR = [25,5,2]
SISTEMA = [50,20,10,5,2,1]

def cambio_monedas(cantidad,sistema):
  '''
     Función que simula el cambio de monedas
     Args:
       cantidad: cantidad de dinero a cambiar
       sistema: sistema de monedas

  '''
  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(116,SISTEMA_ERROR)
cambio_monedas(116,SISTEMA)
```

```
No es posible encontrar solucion
```

Out[ ]:
```
[2, 0, 1, 1, 0, 1]
```

Ejercicio 3 N Reinas

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

#Verifica que en la solución parcial no hay amenzas entre reinas
################################################################
def es_prometedora(SOLUCION,etapa):
################################################################
  #print(SOLUCION)
  #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))
      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)
      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]
[4, 1, 5, 8, 6, 3, 7, 2]
[4, 2, 5, 8, 6, 1, 3, 7]
[4, 2, 7, 3, 6, 8, 1, 5]
[4, 2, 7, 3, 6, 8, 5, 1]
[4, 2, 7, 5, 1, 8, 6, 3]
[4, 2, 8, 5, 7, 1, 3, 6]
[4, 2, 8, 6, 1, 3, 5, 7]
[4, 6, 1, 5, 2, 8, 3, 7]
[4, 6, 8, 2, 7, 1, 3, 5]
[4, 6, 8, 3, 1, 7, 5, 2]
[4, 7, 1, 8, 5, 2, 6, 3]
[4, 7, 3, 8, 2, 5, 1, 6]
[4, 7, 5, 2, 6, 1, 3, 8]
[4, 7, 5, 3, 1, 6, 8, 2]
[4, 8, 1, 3, 6, 2, 7, 5]
[4, 8, 1, 5, 7, 2, 6, 3]
[4, 8, 5, 3, 1, 7, 2, 6]
[5, 1, 4, 6, 8, 2, 7, 3]
[5, 1, 8, 4, 2, 7, 3, 6]
[5, 1, 8, 6, 3, 7, 2, 4]
[5, 2, 4, 6, 8, 3, 1, 7]
[5, 2, 4, 7, 3, 8, 6, 1]
[5, 2, 6, 1, 7, 4, 8, 3]
[5, 2, 8, 1, 4, 7, 3, 6]
[5, 3, 1, 6, 8, 2, 4, 7]
[5, 3, 1, 7, 2, 8, 6, 4]
[5, 3, 8, 4, 7, 1, 6, 2]
[5, 7, 1, 3, 8, 6, 4, 2]
[5, 7, 1, 4, 2, 8, 6, 3]
[5, 7, 2, 4, 8, 1, 3, 6]
[5, 7, 2, 6, 3, 1, 4, 8]
[5, 7, 2, 6, 3, 1, 8, 4]
[5, 7, 4, 1, 3, 8, 6, 2]
[5, 8, 4, 1, 3, 6, 2, 7]
[5, 8, 4, 1, 7, 2, 6, 3]
[6, 1, 5, 2, 8, 3, 7, 4]
[6, 2, 7, 1, 3, 5, 8, 4]
[6, 2, 7, 1, 4, 8, 5, 3]
[6, 3, 1, 7, 5, 8, 2, 4]
[6, 3, 1, 8, 4, 2, 7, 5]
[6, 3, 1, 8, 5, 2, 4, 7]
[6, 3, 5, 7, 1, 4, 2, 8]
[6, 3, 5, 8, 1, 4, 2, 7]
[6, 3, 7, 2, 4, 8, 1, 5]
[6, 3, 7, 2, 8, 5, 1, 4]
[6, 3, 7, 4, 1, 8, 2, 5]
[6, 4, 1, 5, 8, 2, 7, 3]
[6, 4, 2, 8, 5, 7, 1, 3]
[6, 4, 7, 1, 3, 5, 2, 8]
[6, 4, 7, 1, 8, 2, 5, 3]
[6, 8, 2, 4, 1, 7, 5, 3]
[7, 1, 3, 8, 6, 4, 2, 5]
[7, 2, 4, 1, 8, 5, 3, 6]
[7, 2, 6, 3, 1, 4, 8, 5]
[7, 3, 1, 6, 8, 5, 2, 4]
[7, 3, 8, 2, 5, 1, 6, 4]
[7, 4, 2, 5, 8, 1, 3, 6]
[7, 4, 2, 8, 6, 1, 3, 5]
[7, 5, 3, 1, 6, 8, 2, 4]
[8, 2, 4, 1, 7, 5, 3, 6]
[8, 2, 5, 3, 1, 7, 4, 6]
[8, 3, 1, 6, 2, 5, 7, 4]
[8, 4, 1, 3, 6, 2, 7, 5]
```

In [ ]:
```
escribe_solucion([3, 6, 8, 1, 5, 7, 2, 4])
```

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

In [ ]:
```
#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]
]

#999 se puede sustituir por float("inf")


#Calculo de la matriz de PRECIOS y RUTAS
####################################################################
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 por k
      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, 1,6)
```

```
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:
Out[ ]: ',1,4'
```

Ejercicio individual

Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos

- Lista 1D
- Primer intento: Fuerza bruta
- Segundo intento: Aplicar divide y vencerás

In [ ]:
```
PUNTOS = [9057,12543,343,2,34,7,19264,567,234,1000]

def distancia_fuerza_bruta(puntos):
    distancia = None
    solucion = []
    for i,punto_1 in enumerate(puntos):
        for j, punto_2 in enumerate(puntos):
            if i == j:
                pass
            else:
                if distancia == None:
                    distancia = abs(punto_1-punto_2)
                    solucion = [punto_1,punto_2]
                distancia_temp =  abs(punto_1-punto_2)
                if distancia > distancia_temp:
                    distancia = distancia_temp
                    solucion = [punto_1,punto_2]

    return solucion

solucion = distancia_fuerza_bruta(PUNTOS)
print(solucion)
```

```
[2, 7]
```

In [ ]:
```
PUNTOS = [9057,12543,343,2,34,7,19264,567,234,1000]

def distancia_recursive(puntos,distancia=None):

    puntos.sort()
    if len(puntos) ==2:
        if distancia == None or distancia > abs(puntos[0]-puntos[1]):
            return (abs(puntos[0]-puntos[1]),[puntos[0], puntos[1]])
    else:
        return min(distancia_recursive(puntos[1:], distancia), distancia_recursive(puntos[:-1], distancia))

print(distancia_recursive(PUNTOS))
```

```
(5, [2, 7])
```