Nombre: Jean Canevello

https://colab.research.google.com/drive/1KdGsTbI3pbVcxlExfabt7rZqMVulyGcy?authuser=1#scrollTo=rTiqLuX2ZKuL

https://github.com/jcanevello/AlgoritmosOptimizacion/blob/master/Algoritmos_AG1.ipynb

PROBLEMA 1 | DIVIDE Y VENCERÁS | TORRES DE HANOY

Trasladar una torre de varios de niveles de una posición a otra

```
'''
Return: Secuencia de pasos para mover cada nivel de la torre hacia las
        distintas posiciones
Params:
  N: Cantidad de niveles de la torre
  desde: posición inicial de la torre
  hasta: posición final de la torre
'''
def torres_hanoy(N, desde, hasta):
  if N == 1:
    print("Llevar desde :", str(desde), " hasta ", str(hasta))
  else:
    torres_hanoy(N-1, desde, 6-desde-hasta)
    print("Llevar desde :", str(desde), " hasta ", str(hasta))
    torres_hanoy(N-1, 6-desde-hasta, hasta)
```

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

```
     Llevar desde : 1  hasta  3
     Llevar desde : 1  hasta  2
     Llevar desde : 3  hasta  2
     Llevar desde : 1  hasta  3
     Llevar desde : 2  hasta  1
     Llevar desde : 2  hasta  3
     Llevar desde : 1  hasta  3
```

PROBLEMA 2 | ALGORITMOS VORACES | CAMBIO DE MONEDA

Devolver la menor cantidad de monedas de un valor utilizando el sistema de monedas definido para el problema

Solución utilizando algoritmos voraces

```
'''
Return: Numero de monedas del cambio
Params:
  valor: Valor entero o decimal equivalente al monto que se debe cambiar
                por monedas, ejm: 13
  sistema: Lista de enteros o decimales equivalente al valor de las
                monedas, ejem: [1,4,8]
'''
def cambio_moneda(valor, sistema):
  val_init = valor
  solucion = {}
  for moneda in sorted(sistema, reverse=True):
    #calcula el cociente de la división entre el monto y el valor de la moneda
    nro_moneda = valor // moneda
    solucion[moneda] = nro_moneda
    if nro_moneda > 0:
      valor -= nro_moneda*moneda

  return f'valor:{val_init}, sistema de moneda:{sistema}'


print(cambio_moneda(15, [11,5,1]))
print(cambio_moneda(8, [1,4,6]))
```

```
     valor:15, sistema de moneda:[11, 5, 1]
     valor:8, sistema de moneda:[1, 4, 6]
```

El algoritmo Voráz no es una solución eficiente para el problema de cambio de moneda ya que solo funciona bien para casos limitados, por ejemplo para el caso donde se busca cambiar un valor de 8 con monedas 1, 4 y 6 da como resultado 3 monedas(1 moneda de 6 y dos de 1), sin embargo; existe una solución más optima utilizando dos monedas de 4.

Para obtener los resultados óptimos utilizaremos Programación Dinámica para este problema.

Solución utilizando Programación Dinámica

```
  '''
  Return: Matriz con valores 0 y la primera fila con valores infinitos
  Params:
    n_filas: valor entero, número de filas
    n_columnas: valor entero, número de columnas
  '''
  def crear_matriz(n_filas, n_columnas):

    #creación de la matriz
    m = [[0 for _ in range(n_columnas)] for _ in range(n_filas)]

    #asigna valor de infinito a la primera fila
    for j in range(n_columnas):
      m[0][j] = float('inf')

    return m

  '''
  Return: Numero de monedas del cambio
  Params:
    valor_cambio: Valor entero o decimal equivalente al monto que se debe cambiar
                  por monedas, ejm: 13
    sistema_moneda: Lista de enteros o decimales equivalente al valor de las
                  monedas, ejem: [1,4,8]
  '''
  def cambio_moneda2(valor_cambio, sistema_moneda):
    n_filas = len(sistema_moneda)+1
    n_columnas = valor_cambio +1

    #matriz donde se van a guardar todos los valores calculados
    matriz = crear_matriz(n_filas, n_columnas)

    # lista con valores del 0 hasta n_columnas
    vector_valor = [j for j in range(n_columnas)]

    for i in range(1, n_filas):
      for j in range(1, n_columnas):
        #valor de la moneda elegida es mayor al valor de cambio
        if sistema_moneda[i-1] > vector_valor[j]:
          #se asigna el valor de la fila anterior respecto a la misma columna
          matriz[i][j] = matriz[i-1][j]
        else:
          #se asigna el menor valor entre a y b, siendo:
          #a: el valor de la fila anterior respecto a la misma columna
          #b: el valor de la fila respecto a la posición de la columna c.
          #c: diferencia entre el valor de cambio y el valor de la moneda aumentado en 1
          matriz[i][j] = min(matriz[i-1][j],
                             matriz[i][vector_valor[j]-sistema_moneda[i-1]]+1)

    return f"Cantidad de monedas de cambio de {valor_cambio} con sistema {sistema_moneda}: {matriz[-1][-1]}"


print(cambio_moneda2(8, [1,4,6]))
print(cambio_moneda2(15, [1,5,11]))
```

```
Cantidad de monedas de cambio de 8 con sistema [1, 4, 6]: 2
Cantidad de monedas de cambio de 15 con sistema [1, 5, 11]: 3
```

PROBLEMA 3 | ALGORITMOS DE VUELTA ATRÁS | PROBLEMA DE LAS 4 REINAS

Devolver todas las posiciones posibles de N reinas en un tablero de NxN sin realizar jaque entre ellas

```
'''
Return: Valor booleano que indica si la posición elegida es válida
Params:
  solucion: solucion parcial
  etapa: numero de reinas colocadas en la solucion parcial
'''
def es_prometedora(SOLUCION, etapa):

  #Si la solución tiene dos valores iguales no es válida
  for i in range(etapa+1):
    if SOLUCION.count(SOLUCION[i]) > 1:
      return False

    #valida las diagonales
    for j in range(i+1, etapa + 1):
      if abs(i-j) == abs(SOLUCION[i] - SOLUCION[j]) :
        return False

  return True
```

```
  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='')

    print('\n')

  '''
  Return: Listas de posibles soluciones
  Params:
    N: Cantidad de reias en el juego
    solucion: solución parcial
    etapa: Número de reinas colocadas en la solución parcial
  '''
  def reinas(N, solucion=[], etapa=0):

    #inicia con una solución de ceros
    if len(solucion) == 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(4, solucion=[], etapa=0)
       [2, 4, 1, 3]
       [3, 1, 4, 2]


  escribe_solucion([2, 4, 1, 3])
  escribe_solucion([3, 1, 4, 2])


        -  -  X  -
        X  -  -  -
        -  -  -  X
        -  X  -  -


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

PROBLEMA 4 | PROGRAMACIÓN DINÁMICA | VIAJE POR EL RÍO

Devolver la menor tarifa para ir del punto A al punto B

```
  '''
  Return: Dos listas de precio y ruta
  Params:
    tarifa: matriz de precios entre nodo y nodo
  '''
  def precios(tarifas):
    #Total de nodos
    n = len(tarifas[0])

    #creación de la tabla de precios
    precios = [[float('inf')]*n for i in [float('inf')]*n]
    ruta = [['']*n for i in ['']*n]

    #recorre hasta la penultima fila porque los nodos no son bidireccional
    for i in range(n-1):
      #empieza de i+1 porque los nodos no son cíclicos
      for j in range(i+1, n):
        precio_min = tarifas[i][j]
        ruta[i][j] = i

        #calcula el menor precio hasta el nodo j
        for k in range(i,j):
```

```
            if precios[i][k] + tarifas[k][j] < precio_min:
              precio_min = min(precio_min, precios[i][k] + tarifas[k][j])
              ruta[i][j] = k
            precios[i][j] = precio_min

      return precios, ruta

    def calcular_ruta(ruta, desde, hasta):
      if desde == hasta:
        return desde
      else:
        return str(calcular_ruta(ruta, desde, ruta[desde][hasta])) + ',' + str(ruta[desde][hasta])


    def calcular_ruta2(m_ruta, desde, hasta):

      ruta = [hasta]
      while desde != hasta:
        nodo_ant = m_ruta[desde][hasta]
        ruta.insert(0, nodo_ant)
        hasta = nodo_ant

      return ','.join(map(str, ruta))


    tarifas = [
        [0,5,4,3,float('inf'),float('inf'),float('inf')],
        [float('inf'),0,float('inf'),2,3,float('inf'),11],
        [float('inf'),float('inf'),0,1,float('inf'),4,10],
        [float('inf'),float('inf'),float('inf'),0,5,6,9],
        [float('inf'),float('inf'),float('inf'),float('inf'),0,float('inf'),4],
        [float('inf'),float('inf'),float('inf'),float('inf'),float('inf'),float('inf'),0]
    ]

    precios, ruta = precios(tarifas)
    print(f'La ruta de menor costo es: {calcular_ruta2(ruta, 0, 6)}')

         La ruta de menor costo es: 0,2,5,6
```

PROBLEMA 5 | PUNTOS MÁS CERCANOS

Devolver el par de puntos más cercanos de una lista

Algoritmo por Fuerza Bruta

```
import random

'''
Return: Par de números más cercanos
params: Lista de números a evaluar
'''
def puntos_cercanos_fb(puntos):
  tamanoLista = len(puntos)
  disMin = float("inf")
  p1 = 0
  p2 = 0

  for i in range(tamanoLista):
    for j in range(i+1, tamanoLista):
      dis = abs(puntos[i] - puntos[j])

      if dis < disMin:
        disMin = dis
        p1 = puntos[i]
        p2 = puntos[j]

  return p1, p2

lista_1d = random.sample(range(0,10000),10)
print(f'Los puntos más cercanos son:{puntos_cercanos_fb(lista_1d)}')


     Los puntos más cercanos son:(7443, 7470)
```

Algoritmo de Divide y Vencerás para plano en 2D

```
import random
import math

'''
```

```
    Return: lista de puntos ordenados por eje x o y
    Params:
      puntos_2d: lista de puntos en el plano 2D
      eje_x: True si el orden es en el eje x y False para eje y
    '''
    def ordenar_por_eje(puntos_2d, eje_x = True):
      eje = 0 if eje_x else 1
      for _ in range(1, len(puntos_2d)):
        for i in range(0, len(puntos_2d)-1):
          if puntos_2d[i][eje] > puntos_2d[i+1][eje]:
              aux = puntos_2d[i+1]
              puntos_2d[i+1] = puntos_2d[i]
              puntos_2d[i] = aux

      return puntos_2d

    '''
    Return: Distancia entre dos puntos
    Params:
      punto_1: tupla de coordenadas x,y en el plano 2d
      punto_2: tupla de coordenadas x,y en el plano 2d
    '''
    def distancia_euclidiana_2d(punto_1, punto_2):
      return math.sqrt(math.pow(punto_2[0]-punto_1[0], 2) + math.pow(punto_2[1] - punto_1[1], 2))

    '''
    Return: Menor distancia entre una lista de 2 o 3 puntos
    Params:
      list_puntos: lista de coordenadas x,y en el plano 2d
    '''
    def minima_distancia(list_puntos):

      dis_1 = distancia_euclidiana_2d(list_puntos[0], list_puntos[1])
      dis_2 = distancia_euclidiana_2d(list_puntos[0], list_puntos[2]) if len(list_puntos) == 3 else float('inf')
      dis_3 = distancia_euclidiana_2d(list_puntos[1], list_puntos[2]) if len(list_puntos) == 3 else float('inf')
      dis_min = min(dis_1, dis_2, dis_3)

      if dis_min == dis_1 :
        return dis_min, list_puntos[0], list_puntos[1]
      if dis_min == dis_2 :
        return dis_min, list_puntos[0], list_puntos[2]
      if dis_min == dis_3 :
        return dis_min, list_puntos[1], list_puntos[2]

    '''
    Return: Menor distancia y el par de puntos que lo conforma
    Params:
      list_puntos: lista de puntos en plano 2D
    '''

    def puntos_mas_cercanos(list_puntos):
      ctd_puntos = len(list_puntos)
      pi = 0
      pd = 0

      if ctd_puntos <= 3:
        min_distancia, pi, pd = minima_distancia(list_puntos)
      else:
        #genera dos sublistas de igual longitud para encontrar de menor distancia
        #en cada sublista utilizando recursividad
        media_ctd_puntos = int(ctd_puntos/2)
        list_puntos_izquierda = list_puntos[:media_ctd_puntos]
        list_puntos_derecha = list_puntos[media_ctd_puntos:]
        min_distancia_izq, pi_1, pi_2 = puntos_mas_cercanos(list_puntos_izquierda)
        min_distancia_der, pd_1, pd_2 = puntos_mas_cercanos(list_puntos_derecha)

        #mínima distancia encontrada entre las sublistas
        min_distancia = min(min_distancia_izq, min_distancia_der)

        if min_distancia == min_distancia_izq:
          pi = pi_1
          pd = pi_2
        else:
          pi = pd_1
          pd = pd_2

        '''
          Bloque para encontrar puntos más cercanos entre la linea media
        '''
        #calcula el rango de valores en el eje x para la lista central
        valor_medio = abs(list_puntos_izquierda[-1][0] + list_puntos_derecha[0][0])/2
        x_min = valor_medio - min_distancia
        x_max = valor_medio + min_distancia

        #selecciona los puntos que se encuentra dentro del rango x
```

```
        list_puntos_medios = []
        for punto in list_puntos:
          if punto[0] >= x_min and punto[0] <= x_max:
            list_puntos_medios.append(punto)

        list_puntos_medios_asc = ordenar_por_eje(list_puntos_medios, eje_x=False)

        #busca una distancia menor al encontrado en el lado izq y der
        max_index = len(list_puntos_medios_asc)-1
        for i in range(len(list_puntos_medios_asc)):
          for k in range(0, 12):
            if (k+i+1) <= max_index:
              distancia = distancia_euclidiana_2d(
                            list_puntos_medios_asc[i],
                            list_puntos_medios_asc[k+i+1]
                          )

              if distancia < min_distancia:
                min_distancia = distancia
                pi = list_puntos_medios_asc[i]
                pd = list_puntos_medios_asc[k+i+1]

      return min_distancia, pi, pd

    lista_2d = [(random.randrange(1, 10000), random.randrange(1, 10000)) for _ in range(1000)]
    lista_puntos_2d = ordenar_por_eje(lista_2d)
    resultado = puntos_mas_cercanos(lista_puntos_2d)
    print(f'La menor distancia es {resultado[0]} con los puntos {resultado[1]} y {resultado[2]}')
```

```
La menor distancia es 12.529964086141668 con los puntos (8513, 7901) y (8519, 7912)
```