Instalación de librerias

```
!pip install requests
import urllib.request

!pip install tsplib95
import tsplib95
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.27.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2023.5.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4)
Requirement already satisfied: tsplib95 in /usr/local/lib/python3.10/dist-packages (0.7.1)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (8.1.3)
Requirement already satisfied: Deprecated~=1.2.9 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (1.2.14)
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (2.8.8)
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (0.8.10)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from Deprecated~=1.2.9->tsplib95) (1.14.1
```

```
import random   #Libreria para generar numeros y listas aleatorias
import copy     #Permite hacer copias de objetos en python: listas, diccionarios,...
import numpy as np
```

Carga de datos del problema

```
#Librerias y carga del problema

#http://elib.zib.de/pub/mp-testdata/tsp/tsplib/
#Documentacion :
  # https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp95.pdf
  # https://tsplib95.readthedocs.io/usage.html
  # https://tsplib95.readthedocs.io/modules.html#module-tsplib95.models

#Matriz de distancias
file = "swiss42.tsp" ;
urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/swiss42.tsp.gz", file + '.gz')
!gzip -d swiss42.tsp.gz     #Descomprimir el fichero de datos

#Objeto de tsplib95 para nuestro problema problema
problem = tsplib95.load(file)

#Nodos
Nodos = list(problem.get_nodes())

#Aristas
Aristas = list(problem.get_edges())
print(Aristas)

#Coordenadas(si estan disponibles en el ficher)
problem.get_display(1)

#Distancia
problem.get_weight(1, 2)
```

```
gzip: swiss42.tsp already exists; do you wish to overwrite (y or n)? ^C
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (0, 1
34
```

Funciones de la Actividad Guiada 3

```
#Funciones de la Actividad Guiada 3
#Se genera una solucion aleatoria con comienzo en en el nodo 0
def crear_solucion(Nodos):
  solucion = [Nodos[0]]
  for n in Nodos[1:] :
    solucion = solucion + [random.choice(list(set(Nodos) - set(solucion)))]
  return solucion
```

```
#Devuelve la distancia entre dos nodos
def distancia(a,b, problem):
  return problem.get_weight(a,b)


#Devuelve la distancia total de una trayectoria/solucion
def distancia_total(solucion, problem):
  distancia_total = 0
  for i in range(len(solucion)-1):
    distancia_total += distancia(solucion[i] ,solucion[i+1] ,  problem)
  return distancia_total + distancia(solucion[len(solucion)-1] ,solucion[0], problem)
```

Funciones Auxiliares

```
#Genera una poblacion inicial de soluciones de tamaño N.
# Puede ser válida la solución aleatoria de la AG3: crear_solucion(Nodos)
def generar_poblacion(Nodos,N):
  poblacion = []
  for i in range(N):
    poblacion+= [crear_solucion(Nodos)]
  return poblacion
```

Haz doble clic (o pulsa Intro) para editar

```
#Evalua la población y devuelve el mejor individuo
def Evaluar_Poblacion(poblacion, problem):
  distancias= []
  for sol in poblacion:
    distancias+= [distancia_total(sol, problem)]
  return poblacion[distancias.index(min(distancias))], min(distancias)


#Funcion de cruce. Recibe una poblacion(lista de soluciones) y devuelve la población ampliada con los hijos.
# Todos los individuos de la población son selecionados para el cruce(si la población es par)
# Podría aplicarse un proceso previo de selección para elegir los individuos que se desea cruzar.
def Cruzar(poblacion, mutacion, problem):
  len_poblacion= len(poblacion)
  poblacion_modified= poblacion.copy()
  random.shuffle(poblacion_modified)
  while poblacion_modified:
    padre_1, padre_2= poblacion_modified.pop(), poblacion_modified.pop()
    hijo_1, hijo_2= Descendencia((padre_1, padre_2), problem, mutacion)
    poblacion.append(hijo_1)
    poblacion.append(hijo_2)
  return poblacion


#Funcion para generar hijos a partir de 2 padres:
# Se elige el metodo de 1-punto de corte pero es posible usar otros n-puntos, uniforme, dependiendo del problema
def Descendencia(padres, problem, mutacion):
  padre_1, padre_2= padres
  # cambia para cada corte para tener soluciones variadas
  n_split= random.randint(0, len(padre_1) - 1)
  # print(n_split)
  # print("padre_1", len(padre_1))
  # print("padre_2", len(padre_2))
  hijo_1, hijo_2= padre_1[:n_split] + padre_2[n_split:], padre_2[:n_split] + padre_1[n_split:]
  # print("hijo_1", len(hijo_1))
  # print("hijo_2", len(hijo_2))
  hijo_1, hijo_2= Mutar(Factibilizar(hijo_1, problem)), Mutar(Factibilizar(hijo_2, problem))

  return hijo_1, hijo_2


if not set([1,2,3,4,5]) - set([1,3,4]):
  print("Factible")


#Para el operador de cruce 1-punto los hijos generados no son soluciones(algunos nodos se repiten y otros no están)
def Factibilizar(solucion,problem):
  # print("Solución: ",solucion)
  if not set(Nodos) ^ set(solucion):
    # print("Factible???")
    # print("Complemento a solución: ",set(Nodos) - set(solucion))
    return solucion
  else:
    # print("No Factible")
    n_puntos_restantes= list(set(Nodos) ^ set(solucion))
    for i in range(len(solucion)):
```

```
      if solucion.count(solucion[i]) > 1:
        solucion[i] = n_puntos_restantes.pop(0)
        # print(solucion)
  # print("Fin")
  # exit()
  return solucion


#Funcion de mutación. Se eligen dos nodos y se intercambia. Se podrian añadir otros operaradores
# Se hace mutaciones mutacion% de las veces
def Mutar(solucion):
  indice_1, indice_2= random.randint(0, len(solucion) - 1), random.randint(0, len(solucion) - 1)
  solucion[indice_1], solucion[indice_2]= solucion[indice_2], solucion[indice_1]
  return solucion


#Funcion de seleccion de la población. Recibe como parametro una poblacion y
# devuelve una poblacion a la que se ha eliminado individuos poco aptos(fitness alto) y para mantener una poblacion estable de N individ
#Se tiene en cuenta el porcentaje elitismo pasado como parametro
# Para los individuos que no son de la elite podríamos usar una selección de ruleta(proporcional a su fitness)
def Seleccionar(problem,poblacion, N, elitismo):
  if len(poblacion) >= N:
    distancia= []
    for solucion in poblacion:
      distancia+= [distancia_total(solucion, problem)]
    distancia, poblacion= np.array(distancia), np.array(poblacion)
    poblacion= poblacion[np.argsort(distancia)]
    poblacion= poblacion[:N]
    return poblacion.tolist()
  else:
    return poblacion
```

Proceso Principal

```
#Funcion principal del algoritmo genetico
##########################################################3
def algoritmo_genetico(problem=problem,N=100,mutacion=.15,elitismo=.1,generaciones=100):
  # problem = datos del problema
  # N = Tamaño de la población
  # mutacion = probabilidad de una mutación
  # elitismo = porcion de la mejor poblacion a mantener
  # generaciones = nº de generaciones a generar para finalizar

  #Genera la poblacion inicial
  Nodos = list(problem.get_nodes())
  poblacion = generar_poblacion(Nodos,N)

  #Inicializamos valores para la mejor solucion
  (mejor_solucion, mejor_distancia) = Evaluar_Poblacion(poblacion, problem)


  #Condicion de parada
  parar = False
  n=0
  #Inciamos el cliclo de generaciones
  while(parar == False) :

    #Cruce de la poblacion(incluye mutación)
    poblacion = Cruzar(poblacion, mutacion, problem)

    #Seleccionamos la población
    poblacion = Seleccionar(problem,poblacion, N, elitismo)

    #Evaluamos la nueva población
    (mejor_solucion, mejor_distancia) = Evaluar_Poblacion(poblacion, problem)

    print("Generacion #", n, "\nLa mejor solución es:" , mejor_solucion, "\ncon distancia " , mejor_distancia, "\n")

    #Numero de generaciones. Criterio de parada
    if n==generaciones:
      parar = True
    n +=1

  return mejor_solucion


sol = algoritmo_genetico(problem=problem,N=500,mutacion=.3,elitismo=.40,generaciones=250)
print(f"Solución: {sol}")
print(f"Distancia: {distancia_total(sol, problem)}")
```

```
La mejor solución es: [4, 3, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 41, 21, 39, 9, 10, 27, 29, 30, 28, 32, 20, 34, 33, 15, 19, 13, 1
con distancia  1877

Generacion # 238
La mejor solución es: [4, 3, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 41, 21, 39, 9, 10, 27, 29, 30, 28, 32, 20, 34, 33, 15, 19, 13, 1
con distancia  1877

Generacion # 239
La mejor solución es: [4, 3, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 41, 21, 39, 9, 10, 27, 29, 30, 28, 32, 20, 34, 33, 15, 19, 13, 1
con distancia  1877

Generacion # 240
La mejor solución es: [4, 3, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 41, 21, 39, 9, 10, 27, 29, 30, 28, 32, 20, 34, 33, 15, 19, 13, 1
con distancia  1877

Generacion # 241
La mejor solución es: [3, 4, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 10, 8, 28, 30, 29, 41, 23, 21, 39, 38, 20, 34, 33, 37, 19, 13, 1
con distancia  1874

Generacion # 242
La mejor solución es: [3, 4, 6, 0, 1, 7, 5, 26, 18, 12, 11, 25, 10, 8, 28, 30, 29, 41, 23, 21, 39, 38, 20, 34, 33, 37, 19, 13, 1
con distancia  1874

Generacion # 243
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 244
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 245
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 246
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 247
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 248
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 249
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Generacion # 250
La mejor solución es: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19
con distancia  1872

Solución: [3, 0, 1, 4, 6, 7, 5, 26, 18, 12, 25, 11, 10, 41, 9, 21, 39, 28, 27, 29, 30, 32, 34, 20, 33, 37, 16, 13, 19, 14, 15, 3
Distancia: 1872
```