

# Actividad Guiada 3 de Algoritmos de Optimizacion

Nombre: Roberto Merchán González

Github: <https://github.com/robertomergon/03MIAR-Algoritmos-de-Optimizacion>

## Carga de librerias

```
In [ ]: #Hacer Llamadas http a paginas de La red  
%pip install requests  
#Modulo para Las instancias del problema del TSP  
%pip install tsplib95
```



Successfully installed Deprecated-1.2.14 networkx-2.8.8 tabulate-0.8.10 tsplib95-0.7.1

# Carga de los datos del problema

```
In [ ]: import urllib.request #Hacer Llamadas http a paginas de la red
import tsplib95      #Modulo para las instancias del problema del TSP
import math          #Modulo de funciones matematicas. Se usa para exp
import random        #Para generar valores aleatorios

#http://elib.zib.de/pub/mp-testdata/tsp/tsplib/
#Documentacion :
# http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf
# https://tsplib95.readthedocs.io/en/stable/pages/usage.html
# https://tsplib95.readthedocs.io/en/v0.6.1/modules.html
# https://pypi.org/project/tsplib95/

#Descargamos el fichero de datos(Matriz de distancias)
file = "swiss42.tsp" ;
urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf", file)
!gzip -d swiss42.tsp.gz      #Descomprimir el fichero de datos

#Coordendas 51-city problem (Christofides/Eilon)
#file = "eil51.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf", file)

#Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
#file = "att48.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf", file)
```

```
In [ ]: #Carga de datos y generación de objeto problem
#####
problem = tsplib95.load(file)

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

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

```
In [ ]: Aristas
```

```
Out[ ]: [(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, 15),  
          (0, 16),  
          (0, 17),  
          (0, 18),  
          (0, 19),  
          (0, 20),  
          (0, 21),  
          (0, 22),  
          (0, 23),  
          (0, 24),  
          (0, 25),  
          (0, 26),  
          (0, 27),  
          (0, 28),  
          (0, 29),  
          (0, 30),  
          (0, 31),  
          (0, 32),  
          (0, 33),  
          (0, 34),  
          (0, 35),  
          (0, 36),  
          (0, 37),  
          (0, 38),  
          (0, 39),  
          (0, 40),  
          (0, 41),  
          (1, 0),  
          (1, 1),  
          (1, 2),  
          (1, 3),  
          (1, 4),  
          (1, 5),  
          (1, 6),  
          (1, 7),  
          (1, 8),  
          (1, 9),  
          (1, 10),  
          (1, 11),  
          (1, 12),  
          (1, 13),  
          (1, 14),  
          (1, 15),  
          (1, 16),  
          (1, 17),
```

(1, 18),  
(1, 19),  
(1, 20),  
(1, 21),  
(1, 22),  
(1, 23),  
(1, 24),  
(1, 25),  
(1, 26),  
(1, 27),  
(1, 28),  
(1, 29),  
(1, 30),  
(1, 31),  
(1, 32),  
(1, 33),  
(1, 34),  
(1, 35),  
(1, 36),  
(1, 37),  
(1, 38),  
(1, 39),  
(1, 40),  
(1, 41),  
(2, 0),  
(2, 1),  
(2, 2),  
(2, 3),  
(2, 4),  
(2, 5),  
(2, 6),  
(2, 7),  
(2, 8),  
(2, 9),  
(2, 10),  
(2, 11),  
(2, 12),  
(2, 13),  
(2, 14),  
(2, 15),  
(2, 16),  
(2, 17),  
(2, 18),  
(2, 19),  
(2, 20),  
(2, 21),  
(2, 22),  
(2, 23),  
(2, 24),  
(2, 25),  
(2, 26),  
(2, 27),  
(2, 28),  
(2, 29),  
(2, 30),  
(2, 31),  
(2, 32),  
(2, 33),  
(2, 34),  
(2, 35),

(2, 36),  
(2, 37),  
(2, 38),  
(2, 39),  
(2, 40),  
(2, 41),  
(3, 0),  
(3, 1),  
(3, 2),  
(3, 3),  
(3, 4),  
(3, 5),  
(3, 6),  
(3, 7),  
(3, 8),  
(3, 9),  
(3, 10),  
(3, 11),  
(3, 12),  
(3, 13),  
(3, 14),  
(3, 15),  
(3, 16),  
(3, 17),  
(3, 18),  
(3, 19),  
(3, 20),  
(3, 21),  
(3, 22),  
(3, 23),  
(3, 24),  
(3, 25),  
(3, 26),  
(3, 27),  
(3, 28),  
(3, 29),  
(3, 30),  
(3, 31),  
(3, 32),  
(3, 33),  
(3, 34),  
(3, 35),  
(3, 36),  
(3, 37),  
(3, 38),  
(3, 39),  
(3, 40),  
(3, 41),  
(4, 0),  
(4, 1),  
(4, 2),  
(4, 3),  
(4, 4),  
(4, 5),  
(4, 6),  
(4, 7),  
(4, 8),  
(4, 9),  
(4, 10),  
(4, 11),

(4, 12),  
(4, 13),  
(4, 14),  
(4, 15),  
(4, 16),  
(4, 17),  
(4, 18),  
(4, 19),  
(4, 20),  
(4, 21),  
(4, 22),  
(4, 23),  
(4, 24),  
(4, 25),  
(4, 26),  
(4, 27),  
(4, 28),  
(4, 29),  
(4, 30),  
(4, 31),  
(4, 32),  
(4, 33),  
(4, 34),  
(4, 35),  
(4, 36),  
(4, 37),  
(4, 38),  
(4, 39),  
(4, 40),  
(4, 41),  
(5, 0),  
(5, 1),  
(5, 2),  
(5, 3),  
(5, 4),  
(5, 5),  
(5, 6),  
(5, 7),  
(5, 8),  
(5, 9),  
(5, 10),  
(5, 11),  
(5, 12),  
(5, 13),  
(5, 14),  
(5, 15),  
(5, 16),  
(5, 17),  
(5, 18),  
(5, 19),  
(5, 20),  
(5, 21),  
(5, 22),  
(5, 23),  
(5, 24),  
(5, 25),  
(5, 26),  
(5, 27),  
(5, 28),  
(5, 29),

(5, 30),  
(5, 31),  
(5, 32),  
(5, 33),  
(5, 34),  
(5, 35),  
(5, 36),  
(5, 37),  
(5, 38),  
(5, 39),  
(5, 40),  
(5, 41),  
(6, 0),  
(6, 1),  
(6, 2),  
(6, 3),  
(6, 4),  
(6, 5),  
(6, 6),  
(6, 7),  
(6, 8),  
(6, 9),  
(6, 10),  
(6, 11),  
(6, 12),  
(6, 13),  
(6, 14),  
(6, 15),  
(6, 16),  
(6, 17),  
(6, 18),  
(6, 19),  
(6, 20),  
(6, 21),  
(6, 22),  
(6, 23),  
(6, 24),  
(6, 25),  
(6, 26),  
(6, 27),  
(6, 28),  
(6, 29),  
(6, 30),  
(6, 31),  
(6, 32),  
(6, 33),  
(6, 34),  
(6, 35),  
(6, 36),  
(6, 37),  
(6, 38),  
(6, 39),  
(6, 40),  
(6, 41),  
(7, 0),  
(7, 1),  
(7, 2),  
(7, 3),  
(7, 4),  
(7, 5),

(7, 6),  
(7, 7),  
(7, 8),  
(7, 9),  
(7, 10),  
(7, 11),  
(7, 12),  
(7, 13),  
(7, 14),  
(7, 15),  
(7, 16),  
(7, 17),  
(7, 18),  
(7, 19),  
(7, 20),  
(7, 21),  
(7, 22),  
(7, 23),  
(7, 24),  
(7, 25),  
(7, 26),  
(7, 27),  
(7, 28),  
(7, 29),  
(7, 30),  
(7, 31),  
(7, 32),  
(7, 33),  
(7, 34),  
(7, 35),  
(7, 36),  
(7, 37),  
(7, 38),  
(7, 39),  
(7, 40),  
(7, 41),  
(8, 0),  
(8, 1),  
(8, 2),  
(8, 3),  
(8, 4),  
(8, 5),  
(8, 6),  
(8, 7),  
(8, 8),  
(8, 9),  
(8, 10),  
(8, 11),  
(8, 12),  
(8, 13),  
(8, 14),  
(8, 15),  
(8, 16),  
(8, 17),  
(8, 18),  
(8, 19),  
(8, 20),  
(8, 21),  
(8, 22),  
(8, 23),

(8, 24),  
(8, 25),  
(8, 26),  
(8, 27),  
(8, 28),  
(8, 29),  
(8, 30),  
(8, 31),  
(8, 32),  
(8, 33),  
(8, 34),  
(8, 35),  
(8, 36),  
(8, 37),  
(8, 38),  
(8, 39),  
(8, 40),  
(8, 41),  
(9, 0),  
(9, 1),  
(9, 2),  
(9, 3),  
(9, 4),  
(9, 5),  
(9, 6),  
(9, 7),  
(9, 8),  
(9, 9),  
(9, 10),  
(9, 11),  
(9, 12),  
(9, 13),  
(9, 14),  
(9, 15),  
(9, 16),  
(9, 17),  
(9, 18),  
(9, 19),  
(9, 20),  
(9, 21),  
(9, 22),  
(9, 23),  
(9, 24),  
(9, 25),  
(9, 26),  
(9, 27),  
(9, 28),  
(9, 29),  
(9, 30),  
(9, 31),  
(9, 32),  
(9, 33),  
(9, 34),  
(9, 35),  
(9, 36),  
(9, 37),  
(9, 38),  
(9, 39),  
(9, 40),  
(9, 41),

(10, 0),  
(10, 1),  
(10, 2),  
(10, 3),  
(10, 4),  
(10, 5),  
(10, 6),  
(10, 7),  
(10, 8),  
(10, 9),  
(10, 10),  
(10, 11),  
(10, 12),  
(10, 13),  
(10, 14),  
(10, 15),  
(10, 16),  
(10, 17),  
(10, 18),  
(10, 19),  
(10, 20),  
(10, 21),  
(10, 22),  
(10, 23),  
(10, 24),  
(10, 25),  
(10, 26),  
(10, 27),  
(10, 28),  
(10, 29),  
(10, 30),  
(10, 31),  
(10, 32),  
(10, 33),  
(10, 34),  
(10, 35),  
(10, 36),  
(10, 37),  
(10, 38),  
(10, 39),  
(10, 40),  
(10, 41),  
(11, 0),  
(11, 1),  
(11, 2),  
(11, 3),  
(11, 4),  
(11, 5),  
(11, 6),  
(11, 7),  
(11, 8),  
(11, 9),  
(11, 10),  
(11, 11),  
(11, 12),  
(11, 13),  
(11, 14),  
(11, 15),  
(11, 16),  
(11, 17),

(11, 18),  
(11, 19),  
(11, 20),  
(11, 21),  
(11, 22),  
(11, 23),  
(11, 24),  
(11, 25),  
(11, 26),  
(11, 27),  
(11, 28),  
(11, 29),  
(11, 30),  
(11, 31),  
(11, 32),  
(11, 33),  
(11, 34),  
(11, 35),  
(11, 36),  
(11, 37),  
(11, 38),  
(11, 39),  
(11, 40),  
(11, 41),  
(12, 0),  
(12, 1),  
(12, 2),  
(12, 3),  
(12, 4),  
(12, 5),  
(12, 6),  
(12, 7),  
(12, 8),  
(12, 9),  
(12, 10),  
(12, 11),  
(12, 12),  
(12, 13),  
(12, 14),  
(12, 15),  
(12, 16),  
(12, 17),  
(12, 18),  
(12, 19),  
(12, 20),  
(12, 21),  
(12, 22),  
(12, 23),  
(12, 24),  
(12, 25),  
(12, 26),  
(12, 27),  
(12, 28),  
(12, 29),  
(12, 30),  
(12, 31),  
(12, 32),  
(12, 33),  
(12, 34),  
(12, 35),

(12, 36),  
(12, 37),  
(12, 38),  
(12, 39),  
(12, 40),  
(12, 41),  
(13, 0),  
(13, 1),  
(13, 2),  
(13, 3),  
(13, 4),  
(13, 5),  
(13, 6),  
(13, 7),  
(13, 8),  
(13, 9),  
(13, 10),  
(13, 11),  
(13, 12),  
(13, 13),  
(13, 14),  
(13, 15),  
(13, 16),  
(13, 17),  
(13, 18),  
(13, 19),  
(13, 20),  
(13, 21),  
(13, 22),  
(13, 23),  
(13, 24),  
(13, 25),  
(13, 26),  
(13, 27),  
(13, 28),  
(13, 29),  
(13, 30),  
(13, 31),  
(13, 32),  
(13, 33),  
(13, 34),  
(13, 35),  
(13, 36),  
(13, 37),  
(13, 38),  
(13, 39),  
(13, 40),  
(13, 41),  
(14, 0),  
(14, 1),  
(14, 2),  
(14, 3),  
(14, 4),  
(14, 5),  
(14, 6),  
(14, 7),  
(14, 8),  
(14, 9),  
(14, 10),  
(14, 11),

(14, 12),  
(14, 13),  
(14, 14),  
(14, 15),  
(14, 16),  
(14, 17),  
(14, 18),  
(14, 19),  
(14, 20),  
(14, 21),  
(14, 22),  
(14, 23),  
(14, 24),  
(14, 25),  
(14, 26),  
(14, 27),  
(14, 28),  
(14, 29),  
(14, 30),  
(14, 31),  
(14, 32),  
(14, 33),  
(14, 34),  
(14, 35),  
(14, 36),  
(14, 37),  
(14, 38),  
(14, 39),  
(14, 40),  
(14, 41),  
(15, 0),  
(15, 1),  
(15, 2),  
(15, 3),  
(15, 4),  
(15, 5),  
(15, 6),  
(15, 7),  
(15, 8),  
(15, 9),  
(15, 10),  
(15, 11),  
(15, 12),  
(15, 13),  
(15, 14),  
(15, 15),  
(15, 16),  
(15, 17),  
(15, 18),  
(15, 19),  
(15, 20),  
(15, 21),  
(15, 22),  
(15, 23),  
(15, 24),  
(15, 25),  
(15, 26),  
(15, 27),  
(15, 28),  
(15, 29),

(15, 30),  
(15, 31),  
(15, 32),  
(15, 33),  
(15, 34),  
(15, 35),  
(15, 36),  
(15, 37),  
(15, 38),  
(15, 39),  
(15, 40),  
(15, 41),  
(16, 0),  
(16, 1),  
(16, 2),  
(16, 3),  
(16, 4),  
(16, 5),  
(16, 6),  
(16, 7),  
(16, 8),  
(16, 9),  
(16, 10),  
(16, 11),  
(16, 12),  
(16, 13),  
(16, 14),  
(16, 15),  
(16, 16),  
(16, 17),  
(16, 18),  
(16, 19),  
(16, 20),  
(16, 21),  
(16, 22),  
(16, 23),  
(16, 24),  
(16, 25),  
(16, 26),  
(16, 27),  
(16, 28),  
(16, 29),  
(16, 30),  
(16, 31),  
(16, 32),  
(16, 33),  
(16, 34),  
(16, 35),  
(16, 36),  
(16, 37),  
(16, 38),  
(16, 39),  
(16, 40),  
(16, 41),  
(17, 0),  
(17, 1),  
(17, 2),  
(17, 3),  
(17, 4),  
(17, 5),

(17, 6),  
(17, 7),  
(17, 8),  
(17, 9),  
(17, 10),  
(17, 11),  
(17, 12),  
(17, 13),  
(17, 14),  
(17, 15),  
(17, 16),  
(17, 17),  
(17, 18),  
(17, 19),  
(17, 20),  
(17, 21),  
(17, 22),  
(17, 23),  
(17, 24),  
(17, 25),  
(17, 26),  
(17, 27),  
(17, 28),  
(17, 29),  
(17, 30),  
(17, 31),  
(17, 32),  
(17, 33),  
(17, 34),  
(17, 35),  
(17, 36),  
(17, 37),  
(17, 38),  
(17, 39),  
(17, 40),  
(17, 41),  
(18, 0),  
(18, 1),  
(18, 2),  
(18, 3),  
(18, 4),  
(18, 5),  
(18, 6),  
(18, 7),  
(18, 8),  
(18, 9),  
(18, 10),  
(18, 11),  
(18, 12),  
(18, 13),  
(18, 14),  
(18, 15),  
(18, 16),  
(18, 17),  
(18, 18),  
(18, 19),  
(18, 20),  
(18, 21),  
(18, 22),  
(18, 23),

(18, 24),  
(18, 25),  
(18, 26),  
(18, 27),  
(18, 28),  
(18, 29),  
(18, 30),  
(18, 31),  
(18, 32),  
(18, 33),  
(18, 34),  
(18, 35),  
(18, 36),  
(18, 37),  
(18, 38),  
(18, 39),  
(18, 40),  
(18, 41),  
(19, 0),  
(19, 1),  
(19, 2),  
(19, 3),  
(19, 4),  
(19, 5),  
(19, 6),  
(19, 7),  
(19, 8),  
(19, 9),  
(19, 10),  
(19, 11),  
(19, 12),  
(19, 13),  
(19, 14),  
(19, 15),  
(19, 16),  
(19, 17),  
(19, 18),  
(19, 19),  
(19, 20),  
(19, 21),  
(19, 22),  
(19, 23),  
(19, 24),  
(19, 25),  
(19, 26),  
(19, 27),  
(19, 28),  
(19, 29),  
(19, 30),  
(19, 31),  
(19, 32),  
(19, 33),  
(19, 34),  
(19, 35),  
(19, 36),  
(19, 37),  
(19, 38),  
(19, 39),  
(19, 40),  
(19, 41),

(20, 0),  
(20, 1),  
(20, 2),  
(20, 3),  
(20, 4),  
(20, 5),  
(20, 6),  
(20, 7),  
(20, 8),  
(20, 9),  
(20, 10),  
(20, 11),  
(20, 12),  
(20, 13),  
(20, 14),  
(20, 15),  
(20, 16),  
(20, 17),  
(20, 18),  
(20, 19),  
(20, 20),  
(20, 21),  
(20, 22),  
(20, 23),  
(20, 24),  
(20, 25),  
(20, 26),  
(20, 27),  
(20, 28),  
(20, 29),  
(20, 30),  
(20, 31),  
(20, 32),  
(20, 33),  
(20, 34),  
(20, 35),  
(20, 36),  
(20, 37),  
(20, 38),  
(20, 39),  
(20, 40),  
(20, 41),  
(21, 0),  
(21, 1),  
(21, 2),  
(21, 3),  
(21, 4),  
(21, 5),  
(21, 6),  
(21, 7),  
(21, 8),  
(21, 9),  
(21, 10),  
(21, 11),  
(21, 12),  
(21, 13),  
(21, 14),  
(21, 15),  
(21, 16),  
(21, 17),

(21, 18),  
(21, 19),  
(21, 20),  
(21, 21),  
(21, 22),  
(21, 23),  
(21, 24),  
(21, 25),  
(21, 26),  
(21, 27),  
(21, 28),  
(21, 29),  
(21, 30),  
(21, 31),  
(21, 32),  
(21, 33),  
(21, 34),  
(21, 35),  
(21, 36),  
(21, 37),  
(21, 38),  
(21, 39),  
(21, 40),  
(21, 41),  
(22, 0),  
(22, 1),  
(22, 2),  
(22, 3),  
(22, 4),  
(22, 5),  
(22, 6),  
(22, 7),  
(22, 8),  
(22, 9),  
(22, 10),  
(22, 11),  
(22, 12),  
(22, 13),  
(22, 14),  
(22, 15),  
(22, 16),  
(22, 17),  
(22, 18),  
(22, 19),  
(22, 20),  
(22, 21),  
(22, 22),  
(22, 23),  
(22, 24),  
(22, 25),  
(22, 26),  
(22, 27),  
(22, 28),  
(22, 29),  
(22, 30),  
(22, 31),  
(22, 32),  
(22, 33),  
(22, 34),  
(22, 35),

```
(22, 36),
(22, 37),
(22, 38),
(22, 39),
(22, 40),
(22, 41),
(23, 0),
(23, 1),
(23, 2),
(23, 3),
(23, 4),
(23, 5),
(23, 6),
(23, 7),
(23, 8),
(23, 9),
(23, 10),
(23, 11),
(23, 12),
(23, 13),
(23, 14),
(23, 15),
(23, 16),
(23, 17),
(23, 18),
(23, 19),
(23, 20),
(23, 21),
(23, 22),
(23, 23),
(23, 24),
(23, 25),
(23, 26),
(23, 27),
(23, 28),
(23, 29),
(23, 30),
(23, 31),
(23, 32),
(23, 33),
[...]
```

```
NOMBRE: swiss42
TIPO: TSP
COMENTARIO: 42 Staedte Schweiz (Fricker)
DIMENSION: 42
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 15 30 23 32 55 33 37 92 114 92 110 96 90 74 76 82 72 78 82 159 122 131 206 112 57 28 43 70 1
15 0 34 23 27 40 19 32 93 117 88 100 87 75 63 67 71 69 62 63 96 164 132 131 212 106 44 33 5
30 34 0 11 18 57 36 65 62 84 64 89 76 93 95 100 104 98 57 88 99 130 100 101 179 86 51 4 18
23 23 11 0 11 48 26 54 70 94 69 75 75 84 84 89 92 89 54 78 99 141 111 109 89 89 11 11 11 54
32 27 18 11 0 40 20 58 67 92 61 78 65 76 83 89 91 95 43 72 110 141 116 105 190 81 34 19 35
55 40 57 48 40 0 23 55 96 123 78 75 36 36 66 66 63 95 34 34 137 174 156 129 224 90 15 59 75
33 19 36 26 20 23 0 45 85 111 75 82 69 60 63 70 71 85 44 52 115 161 136 122 210 91 25 37 54
37 32 65 54 58 55 45 0 124 149 118 126 113 80 42 42 40 40 87 87 94 158 158 163 242 135 65 6
92 93 62 70 67 96 85 124 0 28 29 68 63 122 148 155 156 159 67 129 148 78 80 39 129 46 82 65
114 117 84 94 92 123 111 149 28 0 54 91 88 150 174 181 182 181 95 157 159 50 65 27 102 65 11
92 88 64 69 61 78 75 118 29 54 0 39 34 99 134 142 141 157 44 110 161 103 109 52 154 22 63 6
110 100 89 89 78 75 82 126 68 91 39 0 14 80 129 139 135 167 39 98 187 136 148 81 186 28 61 9
96 87 76 75 65 62 69 113 63 88 34 14 0 72 117 128 124 153 26 88 174 136 142 82 187 32 48 79
90 75 93 84 76 36 60 80 122 150 99 80 72 0 59 71 63 116 56 25 170 201 189 151 252 104 44 95
74 63 95 84 83 56 63 42 148 174 134 129 117 59 0 11 8 63 93 35 135 223 195 184 273 146 71 9
```

```
In [ ]: #Probamos algunas funciones del objeto problem

#Distancia entre nodos
problem.get_weight(0, 1)

#Todas las funciones
#Documentación: https://tsplib95.readthedocs.io/en/v0.6.1/modules.html

#dir(problem)
```

Out[ ]: 15

## Funciones basicas

```
In [ ]: #Funciones basicas
#####
#Se genera una solucion aleatoria con comienzo en el nodo 0
def crear_solucion(Nodos):
    solucion = [Nodos[0]]
    for n in Nodos[1:]:
        solucion = solucion + [random.choice(list(set(Nodos) - set([Nodos[0]])) - set
    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], prc

sol_temporal = crear_solucion(Nodos)

distancia_total(sol_temporal, problem), sol_temporal
```

```
Out[ ]: (3899,
 [0,
  1,
  11,
  29,
  25,
  12,
  9,
  24,
  41,
  40,
  8,
  27,
  36,
  19,
  23,
  14,
  3,
  4,
  5,
  17,
  16,
  22,
  21,
  39,
  10,
  7,
  35,
  32,
  37,
  31,
  2,
  30,
  18,
  20,
  33,
  34,
  6,
  15,
  28,
  26,
  13,
  38])
```

## BUSQUEDA ALEATORIA

```
In [ ]: #####  
# BUSQUEDA ALEATORIA  
#####  
  
def busqueda_aleatoria(problem, N):  
    #N es el numero de iteraciones  
    Nodos = list(problem.get_nodes())  
  
    mejor_solucion = []  
    #mejor_distancia = 10e100  
    mejor_distancia = float('inf')  
    #Inicializamos con un valor  
    #Inicializamos con un valor
```

```

for i in range(N):
    solucion = crear_solucion(Nodos)
    distancia = distancia_total(solucion, problem)
    #Criterio de parada: repetir
    #Genera una solucion aleatoria
    #Calcula el valor objetivo(a)

    if distancia < mejor_distancia:
        mejor_solucion = solucion
        mejor_distancia = distancia
        #Compara con La mejor obtenida

print("Mejor solución: ", mejor_solucion)
print("Distancia : ", mejor_distancia)
return mejor_solucion

#Busqueda aleatoria con 5000 iteraciones
solucion = busqueda_aleatoria(problem, 10000)

```

Mejor solución: [0, 5, 27, 22, 41, 12, 29, 34, 20, 4, 14, 10, 6, 39, 38, 21, 40, 24, 7, 26, 19, 30, 18, 33, 25, 9, 8, 31, 17, 37, 15, 16, 1, 13, 32, 28, 11, 3, 35, 36, 2, 23]  
Distancia : 3759

## BUSQUEDA LOCAL

```

In [ ]: #####
# BUSQUEDA LOCAL
#####
def genera_vecina(solucion):
    #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos
    #Se puede modificar para aplicar otros generadores distintos que 2-opt
    #print(solucion)
    mejor_solucion = []
    mejor_distancia = 10e100
    for i in range(1,len(solucion)-1):           #Recorremos todos Los nodos en bucle
        for j in range(i+1, len(solucion)):

            #Se genera una nueva solución intercambiando Los dos nodos i,j:
            # (usamos el operador + que para listas en python Las concatena) : ej.: [vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]]

            #Se evalua la nueva solución ...
            distancia_vecina = distancia_total(vecina, problem)

            #... para guardarla si mejora Las anteriores
            if distancia_vecina <= mejor_distancia:
                mejor_distancia = distancia_vecina
                mejor_solucion = vecina
    return mejor_solucion

#solucion = [1, 47, 13, 41, 40, 19, 42, 44, 37, 5, 22, 28, 3, 2, 29, 21, 50, 34,
print("Distancia Solucion Incial:", distancia_total(solucion, problem))

nueva_solucion = genera_vecina(solucion)
print("Distancia Mejor Solucion Local:", distancia_total(nueva_solucion, problem))

```

Distancia Solucion Inicial: 3759  
 Distancia Mejor Solucion Local: 3490

```
In [ ]: #Busqueda Local:
# - Sobre el operador de vecindad 2-opt(funcion genera_vecina)
# - Sin criterio de parada, se para cuando no es posible mejorar.
def busqueda_local(problem):
    mejor_solucion = []

    #Generar una solucion inicial de referencia(aleatoria)
    solucion_referencia = crear_solucion(Nodos)
    mejor_distancia = distancia_total(solucion_referencia, problem)

    iteracion=0          #Un contador para saber las iteraciones que hacemos
    while(1):
        iteracion +=1      #Incrementamos el contador
        #print('#',iteracion)

        #Obtenemos la mejor vecina ...
        vecina = genera_vecina(solucion_referencia)

        #... y la evaluamos para ver si mejoramos respecto a lo encontrado hasta el
        distancia_vecina = distancia_total(vecina, problem)

        #Si no mejoramos hay que terminar. Hemos llegado a un minimo Local(según nuevo criterio)
        if distancia_vecina < mejor_distancia:
            #mejor_solucion = copy.deepcopy(vecina)    #Con copia profunda. Las copias
            mejor_solucion = vecina                    #Guarda la mejor solución encontrada
            mejor_distancia = distancia_vecina

        else:
            print("En la iteracion ", iteracion, ", la mejor solución encontrada es:")
            print("Distancia : ", mejor_distancia)
            return mejor_solucion

        solucion_referencia = vecina

    sol = busqueda_local(problem)
```

En la iteracion 34 , la mejor solución encontrada es: [0, 38, 22, 39, 21, 24, 40, 9, 23, 41, 10, 8, 29, 30, 34, 33, 20, 35, 36, 31, 17, 37, 15, 16, 14, 7, 32, 1, 6, 19, 13, 5, 2, 27, 28, 25, 11, 12, 18, 26, 4, 3]  
 Distancia : 1590

## Ejercicio propuesto

### Modificación de la estructura de entorno

```
In [ ]: # Busqueda local con distintos entornos
def genera_vecina_insert(solucion):
    # Crear una copia de la solución para no modificar la original
    vecina = solucion[:]
    # Seleccionar dos índices aleatorios
    i, j = random.sample(range(len(vecina)), 2)
    # Extraer el nodo en la posición i
    nodo = vecina.pop(i)
    # Insertar el nodo en la posición j
    vecina.insert(j, nodo)
```

```

    return vecina

def genera_vecina_swap(solucion):
    #Generador de soluciones vecinas: Intercambio de dos nodos
    # Crear una copia de la solución para no modificar la original
    vecina = solucion[:]
    # Seleccionar dos índices aleatorios para intercambiar
    i, j = random.sample(range(len(vecina)), 2)
    # Realizar el intercambio
    vecina[i], vecina[j] = vecina[j], vecina[i]
    return vecina

def busqueda_local_entorno_variable(problem, tipo):
    mejor_solucion = []

    #Generar una solución inicial de referencia(aleatoria)
    solucion_referencia = crear_solucion(Nodos)
    mejor_distancia = distancia_total(solucion_referencia, problem)

    iteracion=0           #Un contador para saber las iteraciones que hacemos
    while(1):
        iteracion +=1      #Incrementamos el contador
        #print('#',iteracion)

        #Obtenemos la mejor vecina ...
        vecina = tipo(solucion_referencia)

        #... y la evaluamos para ver si mejoramos respecto a lo encontrado hasta
        distancia_vecina = distancia_total(vecina, problem)

        #Si no mejoramos hay que terminar. Hemos llegado a un mínimo Local(según
        if distancia_vecina < mejor_distancia:
            #mejor_solucion = copy.deepcopy(vecina)      #Con copia profunda. Las c
            mejor_solucion = vecina                      #Guarda la mejor solución
            mejor_distancia = distancia_vecina

        else:
            print("En la iteración ", iteracion, ", la mejor solución encontrada")
            print("Distancia : ", mejor_distancia)
            return mejor_solucion

        solucion_referencia = vecina

    print("Heurística: Insert")
    sol_insert = busqueda_local_entorno_variable(problem, genera_vecina_insert)
    print("Heurística: Swap")
    sol_swap = busqueda_local_entorno_variable(problem, genera_vecina_swap)
    print("Heurística: 2-Opt")
    sol = busqueda_local(problem)

```

Heurística: Insert

En la iteración 2, la mejor solución encontrada es: [0, 23, 10, 24, 25, 18, 37, 33, 11, 36, 31, 35, 21, 1, 14, 22, 26, 17, 16, 2, 41, 3, 12, 20, 27, 29, 38, 9, 34, 6, 4, 32, 8, 15, 39, 19, 7, 40, 30, 5, 13, 28]

Distancia : 4925

Heurística: Swap

En la iteración 3, la mejor solución encontrada es: [40, 3, 17, 26, 0, 30, 41, 37, 8, 39, 13, 18, 34, 5, 9, 7, 14, 31, 28, 38, 24, 2, 16, 22, 12, 27, 21, 4, 11, 15, 29, 25, 35, 33, 19, 10, 32, 6, 1, 20, 36, 23]

Distancia : 5005

Heurística: 2-Opt

En la iteración 31, la mejor solución encontrada es: [0, 27, 32, 31, 35, 36, 17, 37, 15, 18, 41, 23, 34, 33, 20, 1, 6, 5, 19, 13, 26, 4, 3, 2, 22, 38, 7, 16, 14, 39, 21, 24, 40, 25, 11, 12, 10, 8, 9, 29, 30, 28]

Distancia : 2148

Con la utilización de otras heurísticas, observamos que se llega a otras soluciones, de manera más rápida pero no minimizan tanto la distancia

## Permitir movimientos peores respecto a la solución actual (Recocido simulado)

```
In [ ]: def recocido_simulado(problem, temp_inicial, alpha, num_iteraciones):
    solucion_referencia = crear_solucion(Nodos)
    distancia_referencia = distancia_total(solucion_referencia, problem)
    mejor_solucion = []
    mejor_distancia = 10e100
    temp = temp_inicial
    for i in range(num_iteraciones):
        vecina = genera_vecina(solucion_referencia)
        distancia_vecina = distancia_total(vecina, problem)
        if distancia_vecina < mejor_distancia:
            mejor_solucion = vecina
            mejor_distancia = distancia_vecina
        if distancia_vecina < distancia_referencia or probabilidad(temp, abs(distancia_vecina - distancia_referencia)):
            solucion_referencia = vecina
            distancia_referencia = distancia_vecina
        temp *= alpha
    return mejor_solucion

#Función de probabilidad para aceptar peores soluciones
def probabilidad(T,d):
    if random.random() < math.exp( -1*d / T) :
        return True
    else:
        return False

sol = recocido_simulado(problem, 100000000, 0.99, 100)
print(sol)
print(distancia_total(sol, problem))
```

[0, 32, 34, 20, 33, 38, 30, 28, 31, 35, 36, 17, 37, 7, 1, 6, 18, 12, 11, 13, 19, 16, 15, 14, 5, 26, 25, 10, 8, 41, 23, 40, 24, 21, 39, 22, 9, 29, 27, 2, 4, 3]  
1657

## Volver a comenzar con otras soluciones iniciales (Búsqueda Multiarranque)

```
In [ ]: def busqueda_multi_arranque(problem, num_aranques):
    mejor_solucion_global = None
    mejor_distancia_global = float('inf')

    for _ in range(num_aranques):
        solucion_actual = busqueda_local(problem)
        distancia_actual = distancia_total(solucion_actual, problem)

        if distancia_actual < mejor_distancia_global:
            mejor_solucion_global = solucion_actual
            mejor_distancia_global = distancia_actual

    print("Mejor solución global encontrada:", mejor_solucion_global)
    print("Distancia:", mejor_distancia_global)
    return mejor_solucion_global

sol = busqueda_multi_arranque(problem, num_aranques=10)
```

En la iteracion 30 , la mejor solucin encontrada es: [0, 10, 25, 11, 12, 18, 4, 28, 30, 29, 26, 5, 14, 37, 36, 35, 20, 33, 34, 38, 24, 40, 21, 39, 22, 9, 23, 41, 8, 2, 27, 3, 32, 31, 17, 15, 16, 19, 13, 6, 1, 7]  
 Distancia : 1732

En la iteracion 51 , la mejor solucin encontrada es: [0, 32, 30, 29, 28, 2, 27, 3, 1, 7, 17, 37, 15, 4, 39, 21, 24, 40, 23, 41, 25, 11, 12, 6, 31, 36, 35, 20, 3 3, 34, 38, 22, 9, 8, 10, 18, 26, 5, 13, 19, 16, 14]  
 Distancia : 1714

En la iteracion 32 , la mejor solucin encontrada es: [0, 18, 12, 11, 25, 10, 3 1, 36, 35, 20, 33, 34, 9, 23, 41, 8, 29, 30, 28, 2, 27, 22, 38, 32, 3, 6, 26, 5, 13, 19, 14, 16, 15, 37, 17, 7, 1, 4, 21, 40, 24, 39]  
 Distancia : 1925

En la iteracion 38 , la mejor solucin encontrada es: [0, 31, 17, 37, 15, 7, 1, 27, 2, 28, 30, 38, 22, 8, 25, 11, 12, 13, 19, 14, 16, 36, 35, 20, 33, 34, 32, 29, 9, 39, 21, 24, 40, 23, 41, 10, 18, 26, 5, 6, 4, 3]  
 Distancia : 1601

En la iteracion 33 , la mejor solucin encontrada es: [0, 3, 4, 6, 18, 12, 11, 1 3, 19, 14, 7, 1, 27, 2, 28, 30, 22, 40, 24, 38, 34, 33, 20, 35, 36, 32, 8, 23, 4 1, 25, 16, 15, 37, 17, 31, 5, 26, 10, 9, 21, 39, 29]  
 Distancia : 2024

En la iteracion 40 , la mejor solucin encontrada es: [0, 6, 5, 19, 13, 11, 25, 41, 40, 24, 21, 39, 29, 3, 4, 10, 8, 23, 9, 2, 27, 28, 30, 22, 38, 34, 33, 20, 3 5, 36, 31, 17, 18, 12, 26, 14, 16, 15, 37, 7, 1, 32]  
 Distancia : 1792

En la iteracion 33 , la mejor solucin encontrada es: [0, 5, 13, 18, 10, 8, 9, 2 2, 38, 30, 29, 12, 11, 25, 41, 23, 40, 24, 21, 39, 28, 2, 27, 3, 4, 26, 6, 1, 7, 31, 17, 37, 15, 16, 14, 19, 34, 33, 20, 35, 36, 32]  
 Distancia : 1740

En la iteracion 38 , la mejor solucin encontrada es: [0, 27, 28, 30, 29, 10, 2 5, 11, 12, 2, 32, 34, 33, 38, 22, 39, 21, 24, 40, 41, 18, 26, 5, 13, 19, 6, 4, 1, 7, 14, 16, 15, 37, 17, 31, 36, 35, 20, 9, 23, 8, 3]  
 Distancia : 1745

En la iteracion 46 , la mejor solucin encontrada es: [0, 3, 2, 27, 28, 31, 35, 36, 17, 37, 15, 16, 14, 7, 1, 26, 18, 10, 41, 23, 8, 4, 6, 5, 19, 13, 12, 11, 25, 9, 21, 40, 24, 39, 22, 38, 29, 30, 34, 33, 20, 32]  
 Distancia : 1606

En la iteracion 35 , la mejor solucin encontrada es: [0, 19, 13, 5, 6, 20, 33, 34, 32, 1, 4, 3, 27, 2, 28, 22, 38, 7, 37, 15, 16, 14, 26, 18, 12, 11, 29, 30, 9, 8, 10, 25, 41, 23, 21, 40, 24, 39, 31, 35, 36, 17]  
 Distancia : 1930

Mejor solucin global encontrada: [0, 31, 17, 37, 15, 7, 1, 27, 2, 28, 30, 38, 2 2, 8, 25, 11, 12, 13, 19, 14, 16, 36, 35, 20, 33, 34, 32, 29, 9, 39, 21, 24, 40, 23, 41, 10, 18, 26, 5, 6, 4, 3]  
 Distancia: 1601

## SIMULATED ANNEALING

```
In [ ]: #####  
# SIMULATED ANNEALING  
#####  
  
#Generador de 1 solucion vecina 2-opt 100% aleatoria (intercambiar 2 nodos)  
#Mejorable eligiendo otra forma de elegir una vecina.  
def genera_vecina_aleatorio(solucion):  
  
    #Se eligen dos nodos aleatoriamente  
    i,j = sorted(random.sample( range(1,len(solucion)) , 2))
```

```
#Devuelve una nueva solución pero intercambiando los dos nodos elegidos al azar
return solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]

#Función de probabilidad para aceptar peores soluciones
def probabilidad(T,d):
    if random.random() < math.exp( -1*d / T) :
        return True
    else:
        return False

#Función de descenso de temperatura
def bajar_temperatura(T):
    return T*0.99
```

```
In [ ]: def recocido_simulado(problem, TEMPERATURA ):
    #problem = datos del problema
    #T = Temperatura

    solucion_referencia = crear_solucion(Nodos)
    distancia_referencia = distancia_total(solucion_referencia, problem)

    mejor_solucion = []                      #x* del seudocódigo
    mejor_distancia = 10e100                  #F* del seudocódigo

    N=0
    while TEMPERATURA > .0001:
        N+=1
        #Genera una solución vecina
        vecina =genera_vecina_aleatorio(solucion_referencia)

        #Calcula su valor(distancia)
        distancia_vecina = distancia_total(vecina, problem)

        #Si es la mejor solución de todas se guarda(siempre!!!)
        if distancia_vecina < mejor_distancia:
            mejor_solucion = vecina
            mejor_distancia = distancia_vecina

        #Si la nueva vecina es mejor se cambia
        #Si es peor se cambia según una probabilidad que depende de T y delta(distan
        if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(
            #solucion_referencia = copy.deepcopy(vecina)
            solucion_referencia = vecina
            distancia_referencia = distancia_vecina

        #Bajamos la temperatura
        TEMPERATURA = bajar_temperatura(TEMPERATURA)

    print("La mejor solución encontrada es " , end="")
    print(mejor_solucion)
    print("con una distancia total de " , end="")
    print(mejor_distancia)
    return mejor_solucion

sol = recocido_simulado(problem, 10000000)
```

La mejor solución encontrada es [0, 28, 38, 22, 39, 21, 29, 27, 15, 37, 32, 30, 34, 20, 33, 31, 17, 7, 1, 3, 13, 19, 5, 6, 14, 16, 36, 35, 2, 12, 11, 40, 24, 9, 10, 26, 18, 25, 41, 23, 8, 4]  
con una distancia total de 2269