```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import random
from random import sample
from collections import Counter
from itertools import chain, combinations, permutations
get_ipython().run_line_magic('matplotlib', '')
plt.ioff() # Activa Grafica en ventana nueva


# In[316]:


# Distancia de ciudades en diccionarios
cities_distances={'1':{'2':255,'3':512,'4':504,'5':878,'6':975,'7':'ND','8':'ND'
,'9':576,'10':782,

'11':589,'12':670,'13':877,'14':524,'15':473,'16':1520,'17':'ND','18':'ND','Ciud
ad':'Berlin',
                      'cx':13.41053,'cy':52.52437},


'2':{'1':255,'3':534,'4':613,'5':745,'6':'ND','7':'ND','8':'ND','9':365,'10':488
,

'11':493,'12':694,'13':862,'14':744,'15':656,'16':1309,'17':'ND','18':'ND','Ciud
ad':'Hamburg',
                      'cx':9.99302, 'cy':53.55073},


'3':{'1':512,'2':534,'4':190,'5':500,'6':469,'7':862,'8':828,'9':'ND','10':'ND',

'11':250,'12':162,'13':366,'14':534,'15':379,'16':'ND','17':579,'18':968,'Ciudad
':'Stuttgart',
                      'cx':9.17702, 'cy':48.78232},


'4':{'1':504,'2':613,'3':190,'5':685,'6':576,'7':'ND','8':'ND','9':668,'10':603,

'11':'ND','12':242,'13':464,'14':355,'15':201,'16':697,'17':486,'18':838,'Ciudad
':'Munich',
                      'cx':11.57549, 'cy':48.13743},


'5':{'1':878,'2':745,'3':500,'4':685,'6':462,'7':579,'8':744,'9':430,'10':302,
```

```
'11':294,'12':488,'13':410,'14':1034,'15':'ND','16':1460,'17':887,'18':1292,'Ciu
dad':'Paris',
                      'cx':2.3488, 'cy':48.85341},


'6':{'1':975,'2':'ND','3':469,'4':576,'5':462,'7':538,'8':537,'9':'ND','10':567,

'11':461,'12':335,'13':112,'14':'ND','15':'ND','16':749,'17':553,'18':938,'Ciuda
d':'Lyon',
                      'cx':4.84671, 'cy':45.74846},


'7':{'1':'ND','2':'ND','3':862,'4':'ND','5':579,'6':538,'8':244,'9':929,'10':763
,

'11':749,'12':757,'13':545,'14':'ND','15':1199,'16':1105,'17':949,'18':'ND','Ciu
dad':'Bordeaux',
                      'cx':-0.5805, 'cy':44.84044},


'8':{'1':'ND','2':'ND','3':828,'4':'ND','5':744,'6':537,'7':244,'9':1008,'10':83
5,

'11':778,'12':694,'13':469,'14':'ND','15':'ND','16':'ND','17':789,'18':'ND','Ciu
dad':'Toulouse',
                      'cx':1.44367, 'cy':43.60426},


'9':{'1':576,'2':365,'3':'ND','4':668,'5':430,'6':'ND','7':929,'8':1008,'10':173
,

'11':296,'12':613,'13':691,'14':936,'15':'ND','16':1297,'17':1066,'18':'ND','Ciu
dad':'Amsterdam',
                      'cx':4.88969, 'cy':52.37403},


'10':{'1':782,'2':488,'3':'ND','4':603,'5':302,'6':567,'7':763,'8':835,'9':173,

'11':170,'12':492,'13':533,'14':915,'15':770,'16':1501,'17':'ND','18':'ND','Ciud
ad':'Brussels',
                      'cx':4.34878, 'cy':50.85045},


'11':{'1':589,'2':493,'3':250,'4':'ND','5':294,'6':461,'7':749,'8':778,'9':296,'
10':170,

'12':324,'13':401,'14':768,'15':617,'16':'ND','17':'ND','18':'ND','Ciudad':'Luxe
mbourg',
                      'cx':6.13, 'cy':49.61167},


'12':{'1':670,'2':694,'3':162,'4':242,'5':488,'6':335,'7':757,'8':694,'9':613,'1
0':492,
```

```
'11':324,'13':224,'14':592,'15':441,'16':684,'17':453,'18':857,'Ciudad':'Zurich'
,
                    'cx':8.55, 'cy':47.36667},


'13':{'1':877,'2':862,'3':366,'4':464,'5':410,'6':112,'7':545,'8':469,'9':691,'1
0':533,

'11':401,'12':224,'14':804,'15':657,'16':697,'17':485,'18':885,'Ciudad':'Geneva'
,
                    'cx':6.14569, 'cy':46.20222},


'14':{'1':524,'2':744,'3':534,'4':355,'5':1034,'6':'ND','7':'ND','8':'ND','9':93
6,'10':915,

'11':768,'12':592,'13':804,'15':155,'16':765,'17':632,'18':835,'Ciudad':'Vienna'
,
                    'cx':16.37208, 'cy':48.20849},


'15':{'1':473,'2':656,'3':379,'4':201,'5':'ND','6':'ND','7':1199,'8':'ND','9':'N
D','10':770,

'11':617,'12':441,'13':657,'14':155,'16':'ND','17':556,'18':'ND','Ciudad':'Linz'
,
                    'cx':14.28611, 'cy':48.30639},


'16':{'1':1520,'2':1309,'3':'ND','4':697,'5':1460,'6':749,'7':1105,'8':'ND','9':
1297,'10':1501,

'11':'ND','12':684,'13':697,'14':765,'15':'ND','17':231,'18':188,'Ciudad':'Rome'
,
                    'cx':12.51133, 'cy':41.89193},


'17':{'1':'ND','2':'ND','3':579,'4':486,'5':887,'6':553,'7':949,'8':789,'9':1066
,'10':'ND',

'11':'ND','12':453,'13':485,'14':632,'15':556,'16':231,'18':408,'Ciudad':'Floren
cia',
                    'cx':11.24626, 'cy':43.77925},


'18':{'1':'ND','2':'ND','3':968,'4':838,'5':1292,'6':938,'7':'ND','8':'ND','9':'
ND','10':'ND',

'11':'ND','12':857,'13':885,'14':835,'15':'ND','16':188,'17':408,'Ciudad':'Naple
s',
                    'cx':14.26811, 'cy':40.85216 } }


# # Funcion que calcula las distancias entre ciudades, se adjuntan a sus
```

respectivas cromosomas.

In[18]:

```
# Calcula distancias con diccionarios
def calDist(set_ciudad):
    poblacion=set_ciudad.copy()
    dis_Tot=0
    pob_dis=[]
    for cro in poblacion:
        dis_Tot = cities_distances[str(cro[-1])][str(cro[0])]
        ii=1
        #print(cro)
        for gen in cro:
            if ii <= len(cro) - 1:
                #print(cities_distances[str(gen)][str(cro[ii])])
                dis_Tot = dis_Tot + cities_distances[str(gen)][str(cro[ii])]

            ii = ii + 1
        pob_dis += [(cro,dis_Tot)]

    return pob_dis
```

Funcion VERIFICA VALIDEZ DE CROMOSOMA

In[4]:

```
def validacion(cromosoma_x):
    crom_x= cromosoma_x.copy()
    no_NDS= []
    no_NDS+= [cities_distances[str(crom_x[-1])][str(crom_x[0])]]

    ii= 1
    for gen in crom_x:
        if ii <= len(crom_x) - 1:
            no_NDS+= [cities_distances[str(gen)][str(crom_x[ii])]]
        ii+= 1

    return no_NDS
```

Funcion CREACION DE CROMOSOMA VALIDO

In[5]:

```
def crom_validos(cities):
    ciudades=cities.copy()
    valCrom= True
    control=0
    while valCrom:
```

```
        crom_new= random.sample(citKey,len(citKey))

        crom_nuevo= validacion(crom_new)
        if not 'ND' in crom_nuevo:
            return crom_new

        control+=1
    return crom_new


# # Funcion MUTACION SCRAMBLER

# In[171]:


def Mut_Scrambler(selec_muta):
    mutables = selec_muta.copy()
    cromosomas_mutados=[]

    #======================== Seccion que corta, muta e inserta nueva
combinacion de genes
    for mutacion in mutables:
        mutnovalido= True
        while mutnovalido:                 # <----- CICLO SE MANTIENE HASTA
ENCONTRAR CROMOSOMAS MUTADOS VALIDOS
            size_cut = np.random.randint(1,6)    # SE SELECCIONA TAMAÑO DE CORTE
E INDICE DE CORTE CADA CICLO PARA ENCONTRAR VALIDOS
            lisinrange = len(selec_muta[0])-size_cut
            cuts_in_mutacion = np.random.randint(0,lisinrange)

            secc_a_mutar=mutacion[cuts_in_mutacion:cuts_in_mutacion+size_cut]
            ################################ seccion que pasara los cromosomas
no seleccionados para mutar
            secc_inmuta = []
            for gen in mutacion:
                    if gen not in secc_a_mutar:
                        secc_inmuta += [gen]
            ######################### Seccion que muta la lista mutable mediante
cambio de posicion de sus elementos
            cambio_pos=secc_a_mutar.copy()
            cambio_pos += [cambio_pos.pop(0)]

            num = np.random.randint(0,len(secc_inmuta)+1)

            molt= np.insert(secc_inmuta,num,cambio_pos)
            crom_molt=[x for x in molt]
            ##################################################
            molt_nuevo1= validacion(crom_molt.copy())

            if not 'ND' in molt_nuevo1:
                cromosomas_mutados += [crom_molt]
                mutnovalido= False
                #break
```

```
            #print('A buscar otra mutación valida')
            ##################################################

    return cromosomas_mutados


# # Funcion que realiza busqueda de genes faltantes para PMX.

# In[28]:


def busquedaGen(gen,bandera,cromos1,cromos2,new1,new2):
############################################ Para buscar genes faltantes para el
primer newGen
    if bandera ==0:                         # Bandera indica que gen complementa
        val1=cromos1[gen[1]]                # Bandera = 0, busca para gen 1
        valordado=True                      # Bandera = 1, busca para gen 2
        iteracion=0
        while valordado==True:              # Mientra no salga de la busqueda
            if iteracion ==10:              # Control en caso de error en
busqueda
                print('Error!!!:  Iteracion alcanzo 10')
                break
            if val1 not in new1:            # Primero toma valor del indice del
primer 0 en cromosoma
                new1[gen[1]]= val1          # se asigna valor en caso de que no
este
                return new1
            if val1 in new1:
                ind=new1.index(val1)      # se obtiene indice para buscar su
contraparte
            ind
            val1=new2[ind]             #se obtiene nuevo valor para comprobar si
ya esta
            val1
            iteracion=iteracion+1
    ########################################## para buscar genes faltantes para
el segundo newGen
    if bandera ==1:
        val1=cromos2[gen[1]]
        valordado=True
        iteracion=0
        while valordado==True:
            if iteracion ==10:
                print('Error!!!:  Iteracion alcanzo 10')
                break
            if val1 not in new2:
                new2[gen[1]]= val1    #se asigna valor en caso de que no este
                return new2
            if val1 in new2:
                ind=new2.index(val1)      # se obtiene indice para buscar su
contraparte
            ind
            val1=new1[ind]             #se obtiene nuevo valor para comprobar si
```

```
ya esta
            val1
            iteracion=iteracion+1
    ##############################################
    return new1, new2


# # Función CRUZA PMX (Partially Mapped Crossover Operator)

# In[41]:


def cruzaPMX(gen1, gen2):
    cromos1 = gen1.copy()
    cromos2 = gen2.copy()
    novalido= True
    flag= 0
    val1, val2= cromos1,cromos2
    while novalido:                      # <------- CICLO SE MANTIENE HASTA
ENCONTRAR CROMOSOMAS VALIDOS
        size_cut= np.random.randint(1,6)            # SE CAMBIA TAMAÑO DE CORTE
E INDICE DE CORTE ALEATORIO CADA CICLO
        cutinrange= len(set_ciudad[0])-size_cut     # PARA ASEGURAR ENCONTRAR
CROMOSOMAS VALIDOS
        cut_index= np.random.randint(0,cutinrange)

        numCeros = len(cromos1)-size_cut
        newGene1 = [0 for x in range(numCeros)]
        newGene2 = [0 for x in range(numCeros)]

        genpru1 = cromos2[cut_index:cut_index+size_cut]
        genpru2 = cromos1[cut_index:cut_index+size_cut]
        newGe1 = np.insert(newGene1,cut_index,genpru1)
        newGe2 = np.insert(newGene2,cut_index,genpru2)
        newGen1 = [x for x in newGe1]
        newGen2 = [x for x in newGe2]

        for cit in cromos1:
            if cit not in newGen1:
                ds= cit
                sd= cromos1.index(cit)
                if sd < cut_index or sd > cut_index + size_cut-1:
                    newGen1[sd] = ds

            if cit not in newGen2:
                ds= cit
                sd= cromos2.index(cit)
                #if sd < 6 or sd > 11:
                if sd < cut_index or sd > cut_index + size_cut-1:
                    newGen2[sd] = ds

        new1=newGen1.copy()
        new2=newGen2.copy()
    #
```

```
================================================================================
    # Seccion REVISION DE GENES FALTANTES PARA SU BUSQUEDA
        index=0
        zerosingen1=[]
        zerosingen2=[]
        for valzero1,valzero2 in zip(newGen1,newGen2):
            if valzero1 == 0:
                zerosingen1+=[(valzero1,index)]
            if valzero2 == 0:
                zerosingen2+=[(valzero2,index)]
            index=index+1
    #
================================================================================
    # Seccion LLAMADA A FUNCION DE BUSQUEDA DE GENES FALTANTES POR MATRIZ DE
CORRESPONDECIA
        descendiente1= new1
        descendiente2= new2
        for ge1, ge2 in zip(zerosingen1,zerosingen2):
            offspring1 = busquedaGen(ge1,0,cromos1,cromos2,new1,new2)
# PUEDE HABER PROBLEMAS SI NO SE LLEVAN LOS CORRESPONDIENTES ARREGLOS PARA !!!!!

            offspring2 = busquedaGen(ge2,1,cromos1,cromos2,new1,new2)
# REALIZAR EL CAMBIO POR CORRESPONDENCIA Y BUSCAR GENES FALTANTES !!!!!!!!!!!!!
            descendiente1= offspring1.copy()
            descendiente2= offspring2.copy()

        crom_nuevo1= validacion(descendiente1.copy())
        crom_nuevo2= validacion(descendiente2.copy())

        if not 'ND' in crom_nuevo1:
            #print('Enceuntra cromosoma 1')
            val1= descendiente1.copy()
            if flag > 0:  # <--------- VALIDACION SI YA SE ENCONTRO CROMOSOMA 2
                novalido= False
                return val1, val2
            flag+= 1

        if not 'ND' in crom_nuevo2:
            #print('Enceuntra cromosoma 2')
            val2= descendiente2.copy()
            if flag > 0:  # <--------- VALIDACION SI YA SE ENCONTRO CROMOSOMA 2
                novalido= False
                return val1, val2
            flag+= 1

        #print('A buscar otro para de cruzados validos')

    return descendiente1, descendiente2


# # Funcion CRUZA XO(ORDER CROSSOVER OPERATOR)

# In[42]:
```

```
def cruza_OX(c1, c2):
    cromos1 = c1.copy()
    cromos2 = c2.copy()
    no_valido= True
    flags= 0
    val1, val2= cromos1,cromos2
    while no_valido:
        size_cut= np.random.randint(1,6)
        cutinrange= len(set_ciudad[0])-size_cut
        cut_index= np.random.randint(0,cutinrange)

        #size_cut = 6
        genpru1 = cromos1[cut_index:cut_index+size_cut]  # Se corta
aleatoriamente seccion de genes en cromosomas seleccionados
        genpru2 = cromos2[cut_index:cut_index+size_cut]

        ######### Se tranfiere cromosoma2 con nuevo orden definido desde donde
termino corte de genes anterior
        start=cut_index+size_cut
        if start==len(cromos2):start=0
        newordgen1=[]
        newordgen2=[]
        for x in range(len(cromos2)):
            newordgen1+=[cromos1[start]]
            newordgen2+=[cromos2[start]]
            start=start+1
            if start == len(cromos2):start=0
        ######### SE ELIMINAN DEL CROMOSOMA 2 y 1 RESPECTIVAMENTE LOS GENES
SELECCIONADOS DEL CROMOSOMA 1 Y 2
        li2=[gen for gen in newordgen2 if gen not in genpru1]
        li1=[gen for gen in newordgen1 if gen not in genpru2]

        newGe1 = np.insert(li2,cut_index,genpru1) # Se reinsertan en seccion de
cromosoma previamente arreglado
        newGe2 = np.insert(li1,cut_index,genpru2) # en donde se quedo el corte
de seccion, aplica para ambos.

        nuevoGen1 = [x for x in newGe1]  # Finalmente se acomodan los nuevos
GENES
        nuevoGen2 = [x for x in newGe2]

        crom_nuevo1= validacion(nuevoGen1.copy())
        crom_nuevo2= validacion(nuevoGen2.copy())

        if not 'ND' in crom_nuevo1:
            #print('Enceuntra cromosoma 1')
            val1= nuevoGen1.copy()
            if flags > 0:               # <---- VALIDACION SI  YA ENCONTRO
CROMOSOMA 2
                no_valido= False
                return val1, val2
            flags+= 1
```

```
        if not 'ND' in crom_nuevo2:
            #print('Enceuntra cromosoma 2')
            val2= nuevoGen2.copy()
            if flags > 0:               # <---- VALIDACION SI  YA ENCONTRO
CROMOSOMA 1
                no_valido= False
                return val1, val2
            flags+= 1

    return val1, val2


# # Funcion SELECCION TORNEO

# In[228]:


def selec_Torneo(pob,Generaciones,tipoC,tipoM): # tipoC: Tipo de Cruza: PMX OR
OX.  tipoM: Tipo de Mutacion: Scramble or Heuristica
    oldGeneracion=pob.copy()
    theBestPath=[]
    Med_Gener=[]
    toleCambio=100
    ########## CONTROL POR GENERACIONES FOR
    for ng in range(Generaciones):
        print('Generacion #: ',ng, 'Distancia Minima:
','{:.2f}'.format(oldGeneracion[0][1]))

        newGen=[]
        indiv=0
        for i in range(len(oldGeneracion)//2):
            gen1= oldGeneracion[indiv][0]
            gen2= oldGeneracion[len(oldGeneracion)-indiv-1][0]
            if tipoC == 0:
                off1,off2 = cruzaPMX(gen1,gen2)  # < ----- Llamada a cruza PMX
            elif tipoC == 1:
                off1,off2 = cruza_OX(gen1,gen2) # <------- Llamada a CRUZA OX
            #print(off1,off2)
            newGen += [off1.copy()]
            newGen += [off2.copy()]
            indiv += 1

        ##################### Seccion evaluacion de nueva Generacion newGen
        pob_dis=calDist(newGen)
        pob_dis=sorted(pob_dis, key=lambda pob_dis: pob_dis[1], reverse=False)
        ############################### Seccion de Seleccion 50 mejores de 2
Generaciones
        newPoblacion=[]                                 # Se producen 100
individuos
        for indiv1, indiv2 in zip(pob_dis, oldGeneracion): # Aqui se mezclan las
primeras
            newPoblacion += [indiv1,indiv2]           # 2 generaciones y me
genera los mas aptos para su cruza
```

```
        newPoblacion=sorted(newPoblacion, key=lambda newPoblacion:
newPoblacion[1], reverse=False)
        pob_Mejor=newPoblacion[:len(newPoblacion)//2]
        #pob_Mejor=newPoblacion[:len(pob)+4]

        ############################ Seccion MUTACION
        numcromosMut = (10*len(pob_Mejor))//100
        mut=[]
        for _ in range(numcromosMut):
            crom=random.choice(pob_Mejor)
            mut += [crom[0]]
            pob_Mejor.remove(crom)
        if tipoM==0:
            cromosMutados=Mut_Scrambler(mut)    # < ------------ FUNCION
MUTACION
#         elif tipoM==1:
#             cromosMutados= Mut_Heuristica(mut) # < ------------ FUNCION
MUTACION

        newPob_dis=[unoe[0] for unoe in pob_Mejor]                 # Esta
seccion obtiene lista de lista sin su distancia para procesar mutacion se guarda
en aptoss

        ########### SECCION que reinserta cromosomas mutados de nuevo a la
poblacion
        for sdd in cromosMutados:
            indice=np.random.randint(0,len(newPob_dis))
            newPob_dis.insert(indice,sdd)
        ################################ Nuevamente se anexa su distancia
        pob_dis2=calDist(newPob_dis)
        pob_dis2=sorted(pob_dis2, key=lambda pob_dis2: pob_dis2[1],
reverse=False)

        ############################### Seccion evaluacion EPSILON
        dife_absold= [ p[1] for p in oldGeneracion]
        Med_Gener+=[dife_absold]
        dife_absnew= [ p[1] for p in pob_dis2]
        dif_absoluta= abs(sum(dife_absold)-sum(dife_absnew))

        dtot=0
        for lw in range(len(oldGeneracion)):                 # Revision de todos
los cromosomas 1x1, cuantos son iguales que
            if oldGeneracion[0][1] == oldGeneracion[lw][1]:  # el de menor
distancia para determinar epsilon
                dtot +=1

        oldless= oldGeneracion[0][1]
        newless= pob_dis2[0][1]
        theBestPath += [oldGeneracion[0]]

        if dtot >= (90*len(oldGeneracion))//100:  # Si 90% cromosomas iguales
TERMINA EL CICLO
            print('Distancia de ruta alcanzada por mayoria de cromosomas,
Genracion alcanzada: ', ng+1)
```

```
            return oldGeneracion, theBestPath, Med_Gener

        if newless <= oldless:
            oldGeneracion = pob_dis2.copy()
        if newless > oldless:
            oldGeneracion = oldGeneracion
        #print('The best of the best of the best:
','{:.2f}'.format(oldGeneracion[0][1]))
        #oldGeneracion = pob_Mejor.copy()

    return oldGeneracion, theBestPath, Med_Gener


# # Funcion SELECCION RANK

# In[225]:


def selec_Rank(poblacion,Generaciones,tipoC,tipoM):
    oldGeneracion= poblacion.copy()
    theBestPath= []
    Med_GenerRank= []
    toleCambio= 50
    for _ in range(Generaciones):
        print('Generacion #: ', _ ,', Distancia Minima:
''{:.2f}'.format(oldGeneracion[0][1]))
        newGen=[]
        indiv=0
        for i in range(len(oldGeneracion)//2):   # Se toma la longitud del
arreglo "apto" para obtener descendientes
            gen1= oldGeneracion[2*indiv][0]
            gen2= oldGeneracion[2*indiv+1][0]
            if tipoC==0:
                off1,off2 = cruzaPMX(gen1,gen2)  # < --------- Llamada a CRUZA
PMX
            elif tipoC==1:
                off1,off2 = cruza_OX(gen1,gen2) # <------------- Llamada a CRUZA
OX
            newGen += [off1]
            newGen += [off2]
            indiv = indiv+1
        ##################### Seccion evaluacion de nueva Generacion newGen
        dis_Tot=0
        pob_dis=[]
        for cro in newGen:
            dis_Tot = cities_distances[str(cro[-1])][str(cro[0])]

            ii=1
            for gen in cro:
                if ii <= len(cro) - 1:
                    dis_Tot = dis_Tot + cities_distances[str(gen)][str(cro[ii
])]
                ii = ii + 1
            pob_dis += [(cro,dis_Tot)]
```

```
        pob_dis=sorted(pob_dis, key=lambda pob_dis: pob_dis[1], reverse=False)

        ############################## Seccion de Seleccion 50 mejores de 2
Generaciones
        newPoblacion=[]                                    # Se producen 100
individuos
        for indiv1, indiv2 in zip(pob_dis, oldGeneracion): # Aqui se mezclan las
primeras
            newPoblacion += [indiv1,indiv2]           # 2 generaciones y me
genera los mas aptos para su cruza

        newPoblacion=sorted(newPoblacion, key=lambda newPoblacion:
newPoblacion[1], reverse=False)
        pob_Mejor=newPoblacion[:len(newPoblacion)//2]
        #pob_Mejor=newPoblacion[:len(poblacion)+4]

        ########### Seccion MUTACION
        numcromosMut = (10*len(pob_Mejor))//100
        mut=[]
        for yh in range(numcromosMut):
            crom=random.choice(pob_Mejor)
            mut += [crom[0]]
            pob_Mejor.remove(crom)

        if tipoM ==0:
            cromosMutados=Mut_Scrambler(mut) # < --------------------------
FUNCION MUTACION
#         elif tipoM ==1:
#             cromosMutados= Mut_Heuristica(mut) # < ---------------------
FUNCION MUTACION

        newPob_dis=[unoe[0] for unoe in pob_Mejor]                 # Esta
seccion obtiene lista de lista sin su distancia para procesar mutacion se guarda
en aptoss

        for sdd in cromosMutados:
            indice=np.random.randint(0,len(newPob_dis))
            newPob_dis.insert(indice,sdd)
        ########### Nuevamente se anexa su distancia
        pob_dis2=calDist(newPob_dis)
        pob_dis2=sorted(pob_dis2, key=lambda pob_dis2: pob_dis2[1],
reverse=False)
        ################################################
        dife_absold= [ p[1] for p in oldGeneracion]
        Med_GenerRank+= [dife_absold]
        dife_absnew= [ p[1] for p in pob_dis2]
        dif_absoluta= abs(sum(dife_absold)-sum(dife_absnew))

        dtot=0
        for lw in range(len(oldGeneracion)):                   # For revisa
cuantos cromosomas son iguales suma 1 si lo es.
            if oldGeneracion[0][1] == oldGeneracion[lw][1]:    # para control
epsilon
                dtot +=1
```

```
        oldless= oldGeneracion[0][1]
        newless= pob_dis2[0][1]

        if dtot >= (90*len(oldGeneracion))//100:  # Si 90% cromosomas iguales
TERMINA EL CICLO
            epsilon= False
            print('Distancia de ruta alcanzada por mayoria de cromosomas:
Termina ejecucion en iteracion ', _)
            return oldGeneracion, theBestPath, Med_GenerRank
        if newless <= oldless:
            oldGeneracion = pob_dis2.copy()
        if newless > oldless:
            oldGeneracion = oldGeneracion
        theBestPath += [oldGeneracion[0]]
        ################################################
        #oldGeneracion = pob_dis2.copy()

    return oldGeneracion, theBestPath,Med_GenerRank


# # Creacion de cromosomas

# In[290]:


ciudades= [cities_distances[key]['Ciudad'] for key in cities_distances]
citKey= [int(key) for key in cities_distances]

set_ciudad=[]
size_Population= 100

valCrom= True
for _ in range(size_Population):
    newCromosoma= crom_validos(citKey)
    set_ciudad+= [newCromosoma]

set_ciudad
# # #Counter(set_ciudad[28])
dist_City=calDist(set_ciudad)
print('Cromosomas validos ordenados por elitismo')
dist_City = sorted(dist_City, key=lambda dist_City: dist_City[1], reverse=False)
for gen in dist_City:
    print(gen[0], '{:.2f}'.format(gen[1]))


# # Main SELECCION TORNEO

# In[291]:


Gener=150
tipoCruza,TipoMuta = 1,0  # CRUZA: 0= PMX; 1= OX,  MUTACION: 0= SCRAMBLER
mejores_Torneo, thePathTorneo, mediaTorneo =
```

```
selec_Torneo(dist_City,Gener,tipoCruza,TipoMuta)
print()
for gens1 in mejores_Torneo:
    print(gens1[0], '{:.2f}'.format(gens1[1]))
print(len(mejores_Torneo))
# for mid in mediaxG:
#     print(mid)
# print(len(mediaxG))


# In[293]:


top_Gener= []
media_Gener= []
gener= []
varRank=[]
g=1
for mid in mediaTorneo:
    top_Gener+= [mid[0]]
    m= sum(mid)/len(mid)
    media_Gener+= [m]
    vrank= sum((xi-m)**2 for xi in mid)/len(mid)
    varRank+= [vrank**1/2]
    gener+= [g]
    g+=1

#print(varRank)
fig, ax = plt.subplots(figsize=(15,10))
ax.plot(gener,top_Gener, 'k', marker='o',label='Mejor por generacion')
ax.plot(gener,media_Gener, 'r', marker='*',label='Media por generacion')
#ax.scatter(gener[len(gener)-1],top_Gener[len(gener)-1], 'b',
marker='*',label='Media por generacion')
plt.title('100 Cromosomas, # Generaciones: {0},
'.format(len(gener)),fontweight="bold",fontsize=18)
ax.legend()
ax.set_xlabel('Generaciones, \nDistancia minima: {0}
Km'.format(top_Gener[len(gener)-1]),font="Arial",fontsize=18)
ax.set_ylabel('Distancia Km',font="Arial",fontsize=18)
plt.grid()
plt.show()


# # Recorrido de ruta mas corta

# In[328]:


ciudades1=mejores_Torneo[0][0]
ciudades2=mejores_Torneo[2][0]
print(ciudades1,ciudades2)
# distancia_total=23861.88
coor1=[]
coor2=[]
```

```
absc1=[]
absc2=[]
city1=[]
city2=[]
for c1,c2 in zip(ciudades1,ciudades2):
    #print(_)
    coor1 += [cities_distances[str(c1)]['cx']]
    absc1 += [cities_distances[str(c1)]['cy']]
    city1 += [cities_distances[str(c1)]['Ciudad']]

    coor2 += [cities_distances[str(c2)]['cx']]
    absc2 += [cities_distances[str(c2)]['cy']]
    city2 += [cities_distances[str(c2)]['Ciudad']]

plt.subplot(1,2,1)
plt.scatter(coor1,absc1,color='g',zorder=2)
plt.scatter(coor1[0],absc1[0],color='r',zorder=3)
plt.scatter(coor1[-1],absc1[-1],color='k',zorder=3)
plt.plot(coor1,absc1,linestyle='solid',color='blue',zorder=1)
plt.title('Ruta Menor: {0}'.format(mejores_Torneo[0][0]))
plt.xlabel('Coordenadas, \nRuta Menor: {0}'.format(mejores_Torneo[0][1]))
plt.ylabel('Abscisas')
#plt.ylim(0,60)
#plt.xlim(0,40)
for i,label in enumerate(city1):
    plt.annotate(label,(coor1[i],absc1[i]))

plt.subplot(1,2,2)
plt.scatter(coor2,absc2,color='g',zorder=2)
plt.scatter(coor2[0],absc2[0],color='r',zorder=3)
plt.scatter(coor2[-1],absc2[-1],color='k',zorder=3)
plt.plot(coor2,absc2,linestyle='solid',color='blue',zorder=1)
plt.title('Ruta Menor: {0}'.format(mejores_Torneo[2][0]))
plt.xlabel('Coordenadas, \nRuta Menor: {0}'.format(mejores_Torneo[2][1]))
plt.ylabel('Abscisas')
#plt.ylim(0,60)
#plt.xlim(0,40)
for i,label in enumerate(city2):
    plt.annotate(label,(coor2[i],absc2[i]))
plt.show()


# # Main SELECCION RANK

# In[294]:


generaciones= 150
tipoCruza,TipoMuta = 1,0   # CRUZA: 0= PMX; 1= OX,  MUTACION: 0= SCRAMBLER
mejores_Rank,thePathRank,mediaRank =
selec_Rank(dist_City,generaciones,tipoCruza,TipoMuta)
for gens1 in mejores_Rank:
    print(gens1[0], '{:.2f}'.format(gens1[1]))
print(len(mejores_Rank))
```

```
# In[295]:


top_Gener= []
media_Gener= []
gener=[]
g=1
for mid in mediaRank:
    top_Gener+= [mid[0]]
    media_Gener+= [sum(mid)/len(mid)]
    gener+= [g]
    g+=1

fig, ax = plt.subplots(figsize=(15,10))
ax.plot(gener,top_Gener, 'k', marker='o',label='Mejor por generacion')
ax.plot(gener,media_Gener, 'r', marker='*',label='Media por generacion')
plt.title('100 Cromosomas, # Generaciones:
{0}'.format(len(gener)),fontweight="bold",fontsize=18)
ax.legend()
ax.set_xlabel('Generaciones, \nDistancia minima: {0}
Km'.format(top_Gener[len(gener)-1]),font="Arial",fontsize=18)
ax.set_ylabel('Distancia Km',font="Arial",fontsize=18)
plt.grid()
plt.show()


# # Recorrido de ruta mas corta

# In[326]:


ciudades1=mejores_Rank[0][0]
ciudades2=mejores_Rank[1][0]
print(ciudades1,ciudades2)
# distancia_total=23861.88
coor1=[]
coor2=[]
absc1=[]
absc2=[]
city1=[]
city2=[]
for c1,c2 in zip(ciudades1,ciudades2):
    #print(_)
    coor1 += [cities_distances[str(c1)]['cx']]
    absc1 += [cities_distances[str(c1)]['cy']]
    city1 += [cities_distances[str(c1)]['Ciudad']]

    coor2 += [cities_distances[str(c2)]['cx']]
    absc2 += [cities_distances[str(c2)]['cy']]
    city2 += [cities_distances[str(c2)]['Ciudad']]

plt.subplot(1,2,1)
```

```
plt.scatter(coor1,absc1,color='g',zorder=2)
plt.scatter(coor1[0],absc1[0],color='r',zorder=3)
plt.scatter(coor1[-1],absc1[-1],color='k',zorder=3)
plt.plot(coor1,absc1,linestyle='solid',color='blue',zorder=1)
plt.title('Ruta Menor: {0}'.format(mejores_Rank[0][0]))
plt.xlabel('Coordenadas, \nRuta Menor: {0}'.format(mejores_Rank[0][1]))
plt.ylabel('Abscisas')
#plt.ylim(0,60)
#plt.xlim(0,40)
for i,label in enumerate(city1):
    plt.annotate(label,(coor1[i],absc1[i]))

plt.subplot(1,2,2)
plt.scatter(coor2,absc2,color='g',zorder=2)
plt.scatter(coor2[0],absc2[0],color='r',zorder=3)
plt.scatter(coor2[-1],absc2[-1],color='k',zorder=3)
plt.plot(coor2,absc2,linestyle='solid',color='blue',zorder=1)
plt.title('Ruta Menor: {0}'.format(mejores_Rank[1][0]))
plt.xlabel('Coordenadas, \nRuta Menor: {0}'.format(mejores_Rank[1][1]))
plt.ylabel('Abscisas')
#plt.ylim(0,60)
#plt.xlim(0,40)
for i,label in enumerate(city2):
    plt.annotate(label,(coor2[i],absc2[i]))
plt.show()
```