15/10/2019

Problema del Viajante: Búsqueda Exhaustiva, Heurística y Aplicación de Algoritmos Genéticos

ALGORITMOS GENÉTICOS 2019 – TRABAJO PRÁCTICO 3

ANTONELLI (44852) – RECALDE (44704) – ROHN (41355)

Índice

Introducción

El Problema del Viajante

Consiste en encontrar una ruta que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajante.
Siendo N ciudades de un territorio, la distancia entre cada ciudad viene dada por la matriz D: NxN, donde d[x,y] representa la distancia que hay entre la ciudad X y la ciudad Y.

Distancias en kilómetros	*Cdad. de Bs. As.*	*Córdoba*	*Corrientes*	*Formosa*	*La Plata*	*La Rioja*	*Mendoza*	*Neuquén*	*Paraná*	*Posadas*	*Rawson*	*Resistencia*	*Río Gallegos*	*S.F.d.V.d. Catamarca*	*S.M. de Tucumán*	*S.S. de Jujuy*	*Salta*	*San Juan*	*San Luis*	*Santa Fe*	*Santa Rosa*	*Sgo. Del Estero*	*Ushuaia*	*Viedma*
Cdad. de Bs. As.		646	792	933	53	986	985	989	375	834	1127	794	2082	979	1080	1334	1282	1005	749	393	579	939	2373	799
Córdoba	646		677	824	698	340	466	907	348	919	1321	669	2281	362	517	809	745	412	293	330	577	401	2618	1047
Corrientes	792	677		157	830	814	1131	1534	500	291	1845	13	2819	691	633	742	719	1039	969	498	1136	535	3131	1527
Formosa	933	824	157		968	927	1269	1690	656	263	1999	161	2974	793	703	750	741	1169	1117	654	1293	629	3284	1681
La Plata	53	698	830	968		1038	1029	1005	427	857	1116	833	2064	1030	1132	1385	1333	1053	795	444	602	991	2350	789
La Rioja	986	340	814	927	1038		427	1063	659	1098	1548	802	2473	149	330	600	533	283	435	640	834	311	2821	1311
Mendoza	985	466	1131	1269	1029	427		676	790	1384	1201	1121	2081	569	756	1023	957	152	235	775	586	713	2435	1019
Neuquén	989	907	1534	1690	1005	1063	676		1053	1709	543	1529	1410	1182	1370	1658	1591	824	643	1049	422	1286	1762	479
Paraná	375	348	500	656	427	659	790	1053		658	1345	498	2320	622	707	959	906	757	574	19	642	566	2635	1030
Posadas	834	919	291	263	857	1098	1384	1709	658		1951	305	2914	980	924	1007	992	1306	1200	664	1293	827	3207	1624
Rawson	1127	1321	1845	1999	1116	1548	1201	543	1345	1951		1843	975	1647	1827	2120	2054	1340	1113	1349	745	1721	1300	327
Resistencia	794	669	13	161	833	802	1121	1529	498	305	1843		2818	678	620	729	706	1029	961	495	1132	523	3130	1526
Río Gallegos	2082	2281	2819	2974	2064	2473	2081	1410	2320	2914	975	2818		2587	2773	3063	2997	2231	2046	2325	1712	2677	359	1294
S.F.d.V.d. Catamarca	979	362	691	793	1030	149	569	1182	622	980	1647	678	2587		189	477	410	430	540	602	915	166	2931	1391
S.M. de Tucumán	1080	517	633	703	1132	330	756	1370	707	924	1827	620	2773	189		293	228	612	727	689	1088	141	3116	1562
S.S. de Jujuy	1334	809	742	750	1385	600	1023	1658	959	1007	2120	729	3063	477	293		67	874	1017	942	1382	414	3408	1855
Salta	1282	745	719	741	1333	533	957	1591	906	992	2054	706	2997	410	228	67		808	950	889	1316	353	3341	1790
San Juan	1005	412	1039	1169	1053	283	152	824	757	1306	1340	1029	2231	430	612	874	808		284	740	686	583	2585	1141
San Luis	749	293	969	1117	795	435	235	643	574	1200	1113	961	2046	540	727	1017	950	284		560	412	643	2392	882
Santa Fe	393	330	498	654	444	640	775	1049	19	664	1349	495	2325	602	689	942	889	740	560		641	547	2641	1035
Santa Rosa	579	577	1136	1293	602	834	586	422	642	1293	745	1132	1712	915	1088	1382	1316	686	412	641		977	2044	477
Sgo. Del Estero	939	401	535	629	991	311	713	1286	566	827	1721	523	2677	166	141	414	353	583	643	547	977		3016	1446
Ushuaia	2373	2618	3131	3284	2350	2821	2435	1762	2635	3207	1300	3130	359	2931	3116	3408	3341	2585	2392	2641	2044	3016		1605
Viedma	799	1047	1527	1681	789	1311	1019	479	1030	1624	327	1526	1294	1391	1562	1855	1790	1141	882	1035	477	1446	1605	

Elaboración propia - Las distancias son en línea recta y han sido calculadas con la utilización de un programa de conversión y transformación de coordenadas.

Ejercicio 1

Resolver el Problema del viajante utilizando una Búsqueda Exhaustiva.

Ejercicio 2

A. Resolver el ejercicio anterior permitiendo ingresar el inicio del recorrido y usando la heuristica: “Desde cada ciudad ir a la ciudad mas cercana no visitada”. Ademas presentar un mapa de la Republica con el recorrido realizado indicando la partida, el recorrido completo y la longitud del trayecto.
B. Buscar el minimo recorrido para visitar todas las capitales de la Republica Argentina siguiendo la heuristica mencionada en el punto A. Motrar el recorrudo y la longitud del trayecto.
C. Hallar la ruta de distancia minima que logre unir todas las capitales de provincias de la Republica Argentina, utilizando un algoritmo genetico.

Código

Código perteneciente a la resolución de todos los ejercicios

BACKEND

app.py

```
1.  #!/usr/bin/env python
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  TP3: TRAVELING SALESMAN PROBLEM - ENUNCIADO:
6.  1. Hallar la ruta de distancia mínima que logre unir todas las capitales de provincias de la Públic
    a Argentina,
7.  utilizando un método exhaustivo. ¿Puede resolver el problema? Justificar de manera teórica.
8.  2. Realizar un programa que cuente con un menú con las siguientes opciones:
9.      a) Permitir ingresar una provincia y hallar la ruta de distancia mínima que logre unir todas las
    capitales de
10.     provincias de la República Argentina partiendo de dicha capital utilizando la siguiente heurístic
    a:
11.     "Desde cada ciudad ir a la ciudad más cercana no visitada". Recordar regresar siempre a la ciudad
     de partida.
12.     Presentar un mapa de la República con el recorrido indicado. Además indicar la ciudad de partida,
     el recorrido
13.     completo y la longitud del trayecto. El programa deberá permitir seleccionar la capital que el us
    uario desee
14.     ingresar como inicio del recorrido.
15.     b) Encontrar el recorrido mínimo para visitar todas las capitales de las provincias de la Públi
    ca Argentina
16.     siguiendo la heurística mencionada en el punto a. Deberá mostrar como salida recorrido y longitud
     del trayecto.
17.     c) Hallar la ruta de distancia mínima que logre unir todas las capitales de provincias de la Repú
    blica Argentina,
18.     utilizando un algoritmo genético.
19. FECHA DE ENTREGA: 30/09/2019
20. --> Genetic-Algorithm TP3 --- V1.0 ---  Created on 3 sep. 2019
21.             Antonelli, Nicolás - Recalde, Alejando - Rohn, Alex
22. """
23.
24. from flask import Flask
25. from flask import render_template
26. from citiesManager import CitiesManager
27.
28. # App Initialization
29. app = Flask(__name__)
30.
31. # Cities Initialization
32. cities = CitiesManager()
33.
34.
35. # Index
36. @app.route('/')
37. def hello_world():
38.     return render_template("index.html")
39.
40.
```

```
# Best Track starting from a Selected City
@app.route('/Map/<root_city_name>')
def show_track_from(root_city_name):
    track = cities.get_best_track_from(root_city_name)

    print(track)

    return render_template("showMap.html", track=track)


# Best Track of all Tracks
@app.route('/Map')
def show_best_track():
    best_track = cities.get_best_track()

    return render_template("showMap.html", track=best_track)


# Best Track Using Genetic Algorithm
@app.route('/Map/Ag')
def show_track_with_ag():
    track_ag = cities.get_track_with_ag()

    return render_template("showMap.html", track=track_ag)


# Main Function
if __name__ == '__main__':
    app.run()
```

citiesManager.py

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

from neighbourCitiesRepo import NeighbourCitiesRepo
from city import City
from cityToVisit import CityToVisit
from population import Population
from chromosome import Chromosome

# Initialization of All Cities
class CitiesManager(object):
    def __init__(self):
        self.citiesRepo = NeighbourCitiesRepo()

        buenos_aires = City(name="Buenos Aires", cities_neig=self.citiesRepo.get_near_cities_dict("Buenos Aires")
                            , lat=-34.6131516, long=-58.3772316,)

        cordoba = City(name="Cordoba", cities_neig=self.citiesRepo.get_near_cities_dict("Cordoba")
                       , lat=-31.4134998, long=-64.1810532)

        corrientes = City(name="Corrientes", cities_neig=self.citiesRepo.get_near_cities_dict("Corrientes")
                          , lat=-27.4806004, long=-58.8340988)

        formosa = City(name="Formosa", cities_neig=self.citiesRepo.get_near_cities_dict("Formosa")
                       , lat=-26.1775303, long=-58.1781387)

        la_plata = City(name="La Plata", cities_neig=self.citiesRepo.get_near_cities_dict("La Plata")

                        , lat=-34.9214516, long=-57.9545288)
```

```
29.
30.         la_rioja = City(name="La Rioja", cities_neig=self.citiesRepo.get_near_cities_dict("La Rioja")
31.                         , lat=-29.4110508, long=-66.8506699)
32.
33.         mendoza = City(name="Mendoza", cities_neig=self.citiesRepo.get_near_cities_dict("Mendoza")
34.                         , lat=-32.8908386, long=-68.8271713)
35.
36.         neuquen = City(name="Neuquen", cities_neig=self.citiesRepo.get_near_cities_dict("Neuquen")
37.                         , lat=-38.9516106, long=-68.0590973)
38.
39.         parana = City(name="Parana", cities_neig=self.citiesRepo.get_near_cities_dict("Parana")
40.                         , lat=-31.7319698, long=-60.5237999)
41.
42.         posadas = City(name="Posadas", cities_neig=self.citiesRepo.get_near_cities_dict("Posadas")
43.                         , lat=-27.3670807, long=-55.89608)
44.
45.         rawson = City(name="Rawson", cities_neig=self.citiesRepo.get_near_cities_dict("Rawson")
46.                         , lat=-43.3001595, long=-65.1022797)
47.
48.         resistencia = City(name="Resistencia", cities_neig=self.citiesRepo.get_near_cities_dict("Resi
    stencia")
49.                         , lat=-27.4605598, long=-58.9838905)
50.
51.         rio_gallegos = City(name="Rio Gallegos", cities_neig=self.citiesRepo.get_near_cities_dict("Ri
    o Gallegos")
52.                         , lat=-51.6226082, long=-69.218132)
53.
54.         sfdvd_catamarca = City(name="S.F.d.V.d. Catamarca", cities_neig=self.citiesRepo.get_near_citi
    es_dict("S.F.d.V.d. Catamarca")
55.                         , lat=-28.4695702, long=-65.7852402)
56.
57.         sm_de_tucuman = City(name="S.M. de Tucuman", cities_neig=self.citiesRepo.get_near_cities_dict
    ("S.M. de Tucuman")
58.                         , lat=-26.8241405, long=-65.2226028)
59.
60.         ss_de_jujuy = City(name="S.S. de Jujuy", cities_neig=self.citiesRepo.get_near_cities_dict("S.
    S. de Jujuy")
61.                         , lat=-24.1945705, long=-65.2971191)
62.
63.         salta = City(name="Salta", cities_neig=self.citiesRepo.get_near_cities_dict("Salta")
64.                         , lat=-24.7859001, long=-65.4116592)
65.
66.         san_juan = City(name="San Juan", cities_neig=self.citiesRepo.get_near_cities_dict("San Juan")
67.                         , lat=-31.5375004, long=-68.5363922)
68.
69.         san_luis = City(name="San Luis", cities_neig=self.citiesRepo.get_near_cities_dict("San Luis")
70.                         , lat=-33.2950096, long=-66.3356323)
71.
72.         santa_fe = City(name="Santa Fe", cities_neig=self.citiesRepo.get_near_cities_dict("Santa Fe")
73.                         , lat=-31.6333294, long=-60.7000008)
74.
75.         santa_rosa = City(name="Santa Rosa", cities_neig=self.citiesRepo.get_near_cities_dict("Santa
    Rosa")
76.                         , lat=-36.6166687, long=-64.2833328)
77.
78.         sgo_del_estero = City(name="Sgo. del Estero", cities_neig=self.citiesRepo.get_near_cities_dic
    t("Sgo. del Estero")
79.                         , lat=-27.7951107, long=-64.2614899)
80.
81.         ushuaia = City(name="Ushuaia", cities_neig=self.citiesRepo.get_near_cities_dict("Ushuaia")
```

```
                        , lat=-54.7999992, long=-68.3000031)

        viedma = City(name="Viedma", cities_neig=self.citiesRepo.get_near_cities_dict("Viedma")
                        , lat=-40.8134499, long=-62.9966812)

        # Dictionary with all City Objects
        self.cities = {
            "Buenos Aires": buenos_aires,
            "Cordoba": cordoba,
            "Corrientes": corrientes,
            "Formosa": formosa,
            "La Plata": la_plata,
            "La Rioja": la_rioja,
            "Mendoza": mendoza,
            "Neuquen": neuquen,
            "Parana": parana,
            "Posadas": posadas,
            "Rawson": rawson,
                   "Resistencia": resistencia,
                   "Rio Gallegos": rio_gallegos,
                   "S.F.d.V.d. Catamarca": sfdvd_catamarca,
                   "S.M. de Tucuman": sm_de_tucuman,
                   "S.S. de Jujuy": ss_de_jujuy,
                   "Salta": salta,
                   "San Juan": san_juan,
                   "San Luis": san_luis,
                   "Santa Fe": santa_fe,
                   "Santa Rosa": santa_rosa,
                   "Sgo. del Estero": sgo_del_estero,
                   "Ushuaia": ushuaia,
                   "Viedma": viedma,
               }
               print("Correct Initialization")

           # Best Track starting with a Selected One
           def get_best_track_from(self, root_city_name):
               print("Llego a get_best_track_from, parametro: " + root_city_name)
               print(self.cities[root_city_name].get_lat())
               # Create a "CityToVisit" object for Every city on the travel
               root_city = CityToVisit(
                   name=self.cities[root_city_name].get_name(),
                   lat=self.cities[root_city_name].get_lat(),
                   long=self.cities[root_city_name].get_lng())

               # All CityToVisit List and Total Travel Distance
               visited_cities = [root_city]
               accumulated_distance = 0

               # The List of Visited Cities must have the same length of all "Cities" list
               while len(visited_cities) <= len(self.cities):
                   last_city = self.cities[visited_cities[-1].get_name()]

                   # On the last Loop: Go back to the Start City
                   if len(visited_cities) == len(self.cities):
                       accumulated_distance = accumulated_distance + last_city.get_distance_to(root_c
   ity)
                       visited_cities.append(root_city)
                   else:
                       # Check that the next nearest city is not in the List of Visited Cities
                       nearest_neig = last_city.get_nearest_neig([c.name for c in visited_cities])
                       # Add the new City on the Visited Cities List
                       visited_cities.append(CityToVisit(
                           name=self.cities[nearest_neig["name"]].get_name(),
                           lat=self.cities[nearest_neig["name"]].get_lat(),
```

```
                            long=self.cities[nearest_neig["name"]].get_lng())
                        )
                        # Update Total Distance
                        accumulated_distance = accumulated_distance + nearest_neig["distance"]

                # https://stackoverflow.com/questions/21411497/flask-jsonify-a-list-of-objects
                # Returns all cities on Visited Cities List (Serialized) and Total Distance
                return {
                    "cities_to_visit": [c.serialize() for c in visited_cities],
                    "accumulated_distance": accumulated_distance
                }

            # Best Track of all Tracks
            def get_best_track(self):
                # Get Best Track for the First existent City
                best_track = self.get_best_track_from(list(self.cities)[0])

                # Try every posible Track and Compare with the First One
                for c in self.cities:
                    track = self.get_best_track_from(c)
                    if track["accumulated_distance"] < best_track["accumulated_distance"]:
                        best_track = track

                return best_track

            # Best Track Using Genetic Algorithm
            def get_track_with_ag(self):
                # ImportantValues
                iterationLimit = 1500  # Population Iterations
                populationSize = 200  # Initial Population Size
                crossoverProb = 0.75  # Probability of CrossOver
                mutationProb = 0.4  # Probability of Mutation

                track_ag = None  # Initialization of Best Track
                km_ag = 0  # Initialization of Total Distance

                # Initialize Chromosome
                Chromosome.setCitiesDict(self.cities)

                # First Population
                pob = Population(populationSize, list(self.cities.keys()), crossoverProb, mutationProb
    )

                # Iterations
                for iterationCount in range(iterationLimit):
                    values = pob.showPopulation(iterationCount)

                    # In the last iteration, the chromosomes population mustn't reproduce
                    if iterationCount < iterationLimit - 1:
                        pob.reproduce(values["ElitChrom"], values["SecondElitChrom"])  # Reproduction
    of Actual Generation
                        print("------------")

                # Last Reproduction Message
                print("Last Generation Reached Correctly")
                print("------------")
                print()
                print()

                # Get the Best Results after last generation
                track_ag, km_ag = pob.getBestTrackAg()

                # Convert track_ag in a citiesToVisit List
                track_ag_cities = []
```

```
                for city_name in track_ag.getRoute():
                    track_ag_cities.append(CityToVisit(name=city_name, lat=self.cities[city_name].get_
    lat(),
                                                       long=self.cities[city_name].get_lng()))

                root_city = track_ag.getRoute()[0]
                track_ag_cities.append(CityToVisit(name=root_city, lat=self.cities[root_city].get_lat(
    ),
                                                   long=self.cities[root_city].get_lng()))

                # Show Final Results
                print("Best Track:", end=" ")
                for i in range(len(track_ag_cities)):
                    if i < len(track_ag_cities) - 1:
                        print(track_ag_cities[i].get_name(), end=", ")
                    else:
                        print(track_ag_cities[len(track_ag_cities)-
    1].get_name(), "(Vuelta al Inicio)")
                print("Total Distance:", km_ag, "km, after", iterationLimit, "generations with", popul
    ationSize, "Chromosomes")

                return {
                    "cities_to_visit": [c.serialize() for c in track_ag_cities],
                    "accumulated_distance": km_ag
                }
```

city.py

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


# Every City for the Cities Manager's Cities List
class City(object):
    def __init__(self, name, cities_neig, lat, long):
        self.name = name
        self.neig = cities_neig
        self.lat = lat
        self.lng = long

    # Shows the Nearest City among all
    def get_nearest_neig(self, visited_cities):
        # List Comprehension (and Lambda Expression) for Nearest City
        nearest_city = min({x for x in self.neig.items() if x[0] not in visited_cities},
                           key=lambda x: x[1])

        return {
            "name": nearest_city[0],
            "distance": nearest_city[1]
        }

    # Get a Distance between the actual City and another one
    def get_distance_to(self, city):
        return self.neig[city.get_name()]

    def get_name(self):
        return self.name

    def get_lat(self):
        return self.lat

    def get_lng(self):
        return self.lng
```

cityToVisit.py

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


# Every City of the Visited List
class CityToVisit(object):
    def __init__(self, name, lat, long):
        self.name = name
        self.lat = lat
        self.lng = long

    def get_name(self):
        return self.name

    def get_lat(self):
        return self.lat

    def get_lng(self):
        return self.lng

    def serialize(self):
        return {
            "name": self.name,
            "lat": self.lat,
            "lng": self.lng
        }
```

neighbourCitiesRepo.py

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


# Dictionary with all the Cities and Distances between all others
class NeighbourCitiesRepo(object):
    def __init__(self):
        # Completar...
        self.cities = {
           "Buenos Aires": {"Cordoba": 646, "Corrientes": 792, "Formosa": 933, "La Plata": 53, "La Rioja": 986,
            "Mendoza": 985, "Neuquen": 989, "Parana": 375, "Posadas": 834, "Rawson": 1127,
                            "Resistencia": 794, "Rio Gallegos": 2082, "S.F.d.V.d. Catamarca": 979,
                            "S.M. de Tucuman": 1080, "S.S. de Jujuy": 1134, "Salta": 1282, "San Juan": 1005,
                            "San Luis": 749, "Santa Fe": 393, "Santa Rosa": 579, "Sgo. del Estero": 939,
                            "Ushuaia": 2373, "Viedma": 799},

           "Cordoba": {"Buenos Aires": 646, "Corrientes": 677, "Formosa": 824, "La Plata": 698, "La Rioja": 340,

                       "Mendoza": 466, "Neuquen": 907, "Parana": 348, "Posadas": 919, "Rawson": 1321,
                       "Resistencia": 669, "Rio Gallegos": 2281, "S.F.d.V.d. Catamarca": 362,
                       "S.M. de Tucuman": 517, "S.S. de Jujuy": 809, "Salta": 745, "San Juan": 412,
                       "San Luis": 293, "Santa Fe": 330, "Santa Rosa": 577, "Sgo. del Estero": 401,
                       "Ushuaia": 2618, "Viedma": 1047},

           "Corrientes": {"Buenos Aires": 792, "Cordoba": 677, "Formosa": 157, "La Plata": 830, "La Rioja": 814,

                          "Mendoza": 1131, "Neuquen": 1534, "Parana": 500, "Posadas": 291, "Rawson": 1845,
                          "Resistencia": 13, "Rio Gallegos": 2819, "S.F.d.V.d. Catamarca": 691,
```

```
                            "S.M. de Tucuman": 633, "S.S. de Jujuy": 742, "Salta": 719, "San Juan": 1039,
                            "San Luis": 969, "Santa Fe": 498, "Santa Rosa": 1136, "Sgo. del Estero": 535,
                            "Ushuaia": 3131, "Viedma": 1527},

            "Formosa": {"Buenos Aires": 933, "Cordoba": 824, "Corrientes": 157, "La Plata": 968, "La Rioja": 927,
                        "Mendoza": 1269, "Neuquen": 1690, "Parana": 656, "Posadas": 263, "Rawson": 1999,
                        "Resistencia": 161, "Rio Gallegos": 2974, "S.F.d.V.d. Catamarca": 793,
                        "S.M. de Tucuman": 703, "S.S. de Jujuy": 750, "Salta": 741, "San Juan": 1169,
                        "San Luis": 1117, "Santa Fe": 654, "Santa Rosa": 1293, "Sgo. del Estero": 629,
                        "Ushuaia": 3284, "Viedma": 1681},

            "La Plata": {"Buenos Aires": 53, "Cordoba": 698, "Corrientes": 830, "Formosa": 968, "La Rioja": 1038,
                         "Mendoza": 1029, "Neuquen": 1005, "Parana": 427, "Posadas": 857, "Rawson": 1116,
                         "Resistencia": 883, "Rio Gallegos": 2064, "S.F.d.V.d. Catamarca": 1030,
                         "S.M. de Tucuman": 1132, "S.S. de Jujuy": 1385, "Salta": 1333, "San Juan": 1053,
                         "San Luis": 795, "Santa Fe": 444, "Santa Rosa": 602, "Sgo. del Estero": 991,
                         "Ushuaia": 2350, "Viedma": 789},

            "La Rioja": {"Buenos Aires": 986, "Cordoba": 340, "Corrientes": 814, "Formosa": 927, "La Plata": 1038,
                         "Mendoza": 427, "Neuquen": 1063, "Parana": 659, "Posadas": 1098, "Rawson": 1548,
                         "Resistencia": 802, "Rio Gallegos": 2473, "S.F.d.V.d. Catamarca": 149,
                         "S.M. de Tucuman": 330, "S.S. de Jujuy": 600, "Salta": 533, "San Juan": 283,
                         "San Luis": 435, "Santa Fe": 640, "Santa Rosa": 834, "Sgo. del Estero": 311,
                         "Ushuaia": 2821, "Viedma": 1311},

            "Mendoza": {"Buenos Aires": 985, "Cordoba": 466, "Corrientes": 1131, "Formosa": 1269, "La Plata": 1029
,
                        "La Rioja": 427, "Neuquen": 676, "Parana": 790, "Posadas": 1384, "Rawson": 1201,
                        "Resistencia": 1121, "Rio Gallegos": 2081, "S.F.d.V.d. Catamarca": 569,
                        "S.M. de Tucuman": 756, "S.S. de Jujuy": 1023, "Salta": 957, "San Juan": 152,
                        "San Luis": 235, "Santa Fe": 775, "Santa Rosa": 586, "Sgo. del Estero": 713,
                        "Ushuaia": 2435, "Viedma": 1019},

            "Neuquen": {"Buenos Aires": 989, "Cordoba": 907, "Corrientes": 1534, "Formosa": 1690, "La Plata": 1005
,
                        "La Rioja": 1063, "Mendoza": 676, "Parana": 1053, "Posadas": 1709, "Rawson": 543,
                        "Resistencia": 1529, "Rio Gallegos": 1410, "S.F.d.V.d. Catamarca": 1182,
                        "S.M. de Tucuman": 1370, "S.S. de Jujuy": 1658, "Salta": 1591, "San Juan": 824,
                        "San Luis": 643, "Santa Fe": 1049, "Santa Rosa": 422, "Sgo. del Estero": 1286,
                        "Ushuaia": 1762, "Viedma": 479},

            "Parana": {"Buenos Aires": 375, "Cordoba": 348, "Corrientes": 500, "Formosa": 656, "La Plata": 427,
                       "La Rioja": 659, "Mendoza": 790, "Neuquen": 1053, "Posadas": 658, "Rawson": 1345,
                       "Resistencia": 498, "Rio Gallegos": 2230, "S.F.d.V.d. Catamarca": 622,
                       "S.M. de Tucuman": 707, "S.S. de Jujuy": 659, "Salta": 906, "San Juan": 757,
                       "San Luis": 574, "Santa Fe": 19, "Santa Rosa": 642, "Sgo. del Estero": 566,
                       "Ushuaia": 2635, "Viedma": 1030},

            "Posadas": {"Buenos Aires": 834, "Cordoba": 919, "Corrientes": 291, "Formosa": 263, "La Plata": 857,
                        "La Rioja": 1098, "Mendoza": 1384, "Neuquen": 1709, "Parana": 658, "Rawson": 1951,
                        "Resistencia": 305, "Rio Gallegos": 2914, "S.F.d.V.d. Catamarca": 980,
                        "S.M. de Tucuman": 924, "S.S. de Jujuy": 1007, "Salta": 992, "San Juan": 1306,
                        "San Luis": 1200, "Santa Fe": 664, "Santa Rosa": 1293, "Sgo. del Estero": 827,
                        "Ushuaia": 3207, "Viedma": 1624},

            "Rawson": {"Buenos Aires": 1127, "Cordoba": 1321, "Corrientes": 1845, "Formosa": 1999, "La Plata": 111
6,
                       "La Rioja": 1548, "Mendoza": 1201, "Neuquen": 543, "Parana": 1345, "Posadas": 1951,
                       "Resistencia": 1843, "Rio Gallegos": 975, "S.F.d.V.d. Catamarca": 1647,
                       "S.M. de Tucuman": 1827, "S.S. de Jujuy": 2120, "Salta": 2054, "San Juan": 1340,
                       "San Luis": 1113, "Santa Fe": 1349, "Santa Rosa": 745, "Sgo. del Estero": 1721,
                       "Ushuaia": 1300, "Viedma": 327},
```

```
86.
87.             "Resistencia": {"Buenos Aires": 794, "Cordoba": 669, "Corrientes": 13, "Formosa": 161, "La Plata": 833
    ,
88.                             "La Rioja": 802,
89.                             "Mendoza": 1121, "Neuquen": 1529, "Parana": 498, "Posadas": 305, "Rawson": 1843,
90.                             "Rio Gallegos": 2818, "S.F.d.V.d. Catamarca": 678,
91.                             "S.M. de Tucuman": 620, "S.S. de Jujuy": 729, "Salta": 706, "San Juan": 1029,
92.                             "San Luis": 961, "Santa Fe": 495, "Santa Rosa": 1132, "Sgo. del Estero": 523,
93.                             "Ushuaia": 3130, "Viedma": 1526},
94.
95.             "Rio Gallegos": {"Buenos Aires": 2082, "Cordoba": 2281, "Corrientes": 2819, "Formosa": 2974,
96.                              "La Plata": 2064, "La Rioja": 2473, "Mendoza": 2081, "Neuquen": 1410, "Parana": 2320,
97.                              "Posadas": 2914, "Rawson": 975, "Resistencia": 2818, "S.F.d.V.d. Catamarca": 2587,
98.                              "S.M. de Tucuman": 2773, "S.S. de Jujuy": 3063, "Salta": 2997, "San Juan": 2231,
99.                              "San Luis": 2046, "Santa Fe": 2325, "Santa Rosa": 1712, "Sgo. del Estero": 2677,
100.                             "Ushuaia": 359, "Viedma": 1294},
101.
102.            "S.F.d.V.d. Catamarca": {"Buenos Aires": 979, "Cordoba": 362, "Corrientes": 691, "Formosa": 793,
103.                                     "La Plata": 1030, "La Rioja": 149, "Mendoza": 569, "Neuquen": 1182, "Parana":
     622,
104.                                     "Posadas": 980, "Rawson": 1647, "Resistencia": 678, "Rio Gallegos": 2587,
105.                                     "S.M. de Tucuman": 189, "S.S. de Jujuy": 477, "Salta": 410, "San Juan": 430,
106.                                     "San Luis": 540, "Santa Fe": 689, "Santa Rosa": 1088, "Sgo. del Estero": 166,
107.                                     "Ushuaia": 3116, "Viedma": 1562},
108.
109.            "S.M. de Tucuman": {"Buenos Aires": 1080, "Cordoba": 517, "Corrientes": 633, "Formosa": 703,
110.                                "La Plata": 1132, "La Rioja": 330, "Mendoza": 756, "Neuquen": 1370, "Parana": 707,
111.                                "Posadas": 924, "Rawson": 1827, "Resistencia": 620, "Rio Gallegos": 2773,
112.                                "S.F.d.V.d. Catamarca": 189, "S.S. de Jujuy": 293, "Salta": 228, "San Juan": 612,
113.                                "San Luis": 727, "Santa Fe": 689, "Santa Rosa": 1088, "Sgo. del Estero": 141,
114.                                "Ushuaia": 3116, "Viedma": 1562},
115.
116.            "S.S. de Jujuy": {"Buenos Aires": 1334, "Cordoba": 809, "Corrientes": 742, "Formosa": 750,
117.                              "La Plata": 1385, "La Rioja": 600, "Mendoza": 1023, "Neuquen": 1658, "Parana": 959,
118.                              "Posadas": 1007, "Rawson": 2120, "Resistencia": 729, "Rio Gallegos": 3063,
119.                              "S.F.d.V.d. Catamarca": 477, "S.M. de Tucuman": 293, "Salta": 67, "San Juan": 874,
120.                              "San Luis": 1017, "Santa Fe": 942, "Santa Rosa": 1382, "Sgo. del Estero": 414,
121.                              "Ushuaia": 3408, "Viedma": 1855},
122.
123.            "Salta": {"Buenos Aires": 1282, "Cordoba": 745, "Corrientes": 719, "Formosa": 741, "La Plata": 1333,
124.                      "La Rioja": 233, "Mendoza": 957, "Neuquen": 1591, "Parana": 906, "Posadas": 992, "Rawson": 2
    054,
125.                      "Resistencia": 706, "Rio Gallegos": 2997, "S.F.d.V.d. Catamarca": 410,
126.                      "S.M. de Tucuman": 228, "S.S. de Jujuy": 67, "San Juan": 808,
127.                      "San Luis": 950, "Santa Fe": 889, "Santa Rosa": 1316, "Sgo. del Estero": 353,
128.                      "Ushuaia": 3341, "Viedma": 1790},
129.
130.            "San Juan": {"Buenos Aires": 1005, "Cordoba": 412, "Corrientes": 1039, "Formosa": 1169, "La Plata": 10
    53,
131.                         "La Rioja": 283, "Mendoza": 152, "Neuquen": 824, "Parana": 757, "Posadas": 1306, "Rawson"
    : 1340,
132.                         "Resistencia": 1029, "Rio Gallegos": 2231, "S.F.d.V.d. Catamarca": 430,
133.                         "S.M. de Tucuman": 612, "S.S. de Jujuy": 874, "Salta": 808,
134.                         "San Luis": 284, "Santa Fe": 740, "Santa Rosa": 686, "Sgo. del Estero": 583,
135.                         "Ushuaia": 2585, "Viedma": 1141},
136.
137.            "San Luis": {"Buenos Aires": 749, "Cordoba": 293, "Corrientes": 969, "Formosa": 1117, "La Plata": 795,
```

```
                        "La Rioja": 435, "Mendoza": 235, "Neuquen": 643, "Parana": 574, "Posadas": 1200, "Rawson": 1113,
                        "Resistencia": 961, "Rio Gallegos": 2046, "S.F.d.V.d. Catamarca": 540,
                        "S.M. de Tucuman": 727, "S.S. de Jujuy": 1017, "Salta": 950, "San Juan": 284,
                        "Santa Fe": 560, "Santa Rosa": 412, "Sgo. del Estero": 643, "Ushuaia": 2392, "Viedma": 882},

            "Santa Fe": {"Buenos Aires": 393, "Cordoba": 330, "Corrientes": 498, "Formosa": 654, "La Plata": 444,
                        "La Rioja": 640,
                        "Mendoza": 775, "Neuquen": 1049, "Parana": 19, "Posadas": 664, "Rawson": 1349,
                        "Resistencia": 495, "Rio Gallegos": 2325, "S.F.d.V.d. Catamarca": 602,
                        "S.M. de Tucuman": 689, "S.S. de Jujuy": 942, "Salta": 889, "San Juan": 740,
                        "San Luis": 560, "Santa Rosa": 641, "Sgo. del Estero": 547,
                        "Ushuaia": 2641, "Viedma": 1035},

            "Santa Rosa": {"Buenos Aires": 579, "Cordoba": 577, "Corrientes": 1136, "Formosa": 1293, "La Plata": 602,
                          "La Rioja": 834,
                          "Mendoza": 586, "Neuquen": 422, "Parana": 642, "Posadas": 1293, "Rawson": 745,
                          "Resistencia": 1132, "Rio Gallegos": 1712, "S.F.d.V.d. Catamarca": 915,
                          "S.M. de Tucuman": 1088, "S.S. de Jujuy": 1382, "Salta": 1316, "San Juan": 686,
                          "San Luis": 412, "Santa Fe": 641, "Sgo. del Estero": 977,
                          "Ushuaia": 2044, "Viedma": 477},

            "Sgo. del Estero": {"Buenos Aires": 979, "Cordoba": 401, "Corrientes": 535, "Formosa": 629, "La Plata": 991,
                               "La Rioja": 311,
                               "Mendoza": 713, "Neuquen": 1286, "Parana": 566, "Posadas": 827, "Rawson": 1721,
                               "Resistencia": 523, "Rio Gallegos": 2677, "S.F.d.V.d. Catamarca": 166,
                               "S.M. de Tucuman": 141, "S.S. de Jujuy": 414, "Salta": 353, "San Juan": 583,
                               "San Luis": 643, "Santa Fe": 547, "Santa Rosa": 977,
                               "Ushuaia": 3016, "Viedma": 1446},

            "Ushuaia": {"Buenos Aires": 2373, "Cordoba": 2618, "Corrientes": 3131, "Formosa": 3284, "La Plata": 2350,
                        "La Rioja": 2821, "Mendoza": 2435, "Neuquen": 1762, "Parana": 2635, "Posadas": 3207,
                        "Rawson": 1300, "Resistencia": 3130, "Rio Gallegos": 359, "S.F.d.V.d. Catamarca": 2931,
                        "S.M. de Tucuman": 3116, "S.S. de Jujuy": 3408, "Salta": 3341, "San Juan": 2585,
                        "San Luis": 2392, "Santa Fe": 2641, "Santa Rosa": 2044, "Sgo. del Estero": 3016,
                        "Viedma": 1605},

            "Viedma": {"Buenos Aires": 799, "Cordoba": 1047, "Corrientes": 1527, "Formosa": 1681, "La Plata": 789,
                       "La Rioja": 1311, "Mendoza": 1019, "Neuquen": 479, "Parana": 1030, "Posadas": 1624, "Rawson": 327,
                       "Resistencia": 1526, "Rio Gallegos": 1294, "S.F.d.V.d. Catamarca": 1391,
                       "S.M. de Tucuman": 1562, "S.S. de Jujuy": 1855, "Salta": 1790, "San Juan": 1141,
                       "San Luis": 882, "Santa Fe": 1035, "Santa Rosa": 477, "Sgo. del Estero": 1446,
                       "Ushuaia": 1605},

                # More 16 Cities like "Santa Fe": {"Buenos Aires": 4, "La Plata": 5, "Cordoba": 13},
                }

            # Get one City element of the Dictionary
            def get_near_cities_dict(self, city):
                return self.cities[city]
```

chormosome.py

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

```

```
4.  import random
5.
6.
7.  class Chromosome(object):
8.
9.      # Class Attribute
10.     CitiesDict = {}
11.
12.     # Constructor / Instance Attributes
13.     def __init__(self, large, cities, newRoute):
14.         # Chromosome's Genes
15.         if newRoute is None:
16.             random.shuffle(cities)
17.             self.route = []
18.             for i in range(len(cities)):
19.                 self.route.append(cities[i])  # Shuffle and then Assignation
20.             print(self.route)
21.         else:
22.             self.route = newRoute
23.         self.large = large
24.         self.objectivePunctuation = 0
25.         self.fitness = 0
26.
27.         # Initialize Objective Function Punctuation
28.         self.setObjectivePunctuation()
29.
30.
31.     @classmethod
32.     def getCitiesDict(cls):
33.         return cls.CitiesDict
34.
35.     @classmethod
36.     def setCitiesDict(cls, citiesDictionary):
37.         cls.CitiesDict = citiesDictionary
38.
39.     # Show All Genes of the Chromosome
40.     def getRoute(self):
41.         return self.route
42.
43.     def getAccumulatedDistance(self):
44.         accumulated_distance = 0
45.         cd = self.getCitiesDict()
46.         city_last = self.route[0]
47.         # Calculate Distance Step by Step comparing the Chromosome's Route with Neighbours on Cities
    Dictionary
48.         for i in range(1, len(self.route) - 2):
49.             city_step = self.route[i]
50.             accumulated_distance += cd[city_last].get_distance_to(cd[city_step])
51.             city_last = city_step
52.         # At the End, go back to the First City
53.         accumulated_distance += cd[self.route[len(self.route) - 1]].get_distance_to(cd[self.route[0]]
    )
54.         return accumulated_distance
55.
56.     def calcObjPunc(self):
57.         return 1 / self.getAccumulatedDistance()
58.
59.     def calcFitness(self, totalObj):  # Reverse Score (Little Distances gets better Fitness)
60.         # self.fitness = 1000 / self.getObjectivePunctuation()
61.         self.fitness = (self.getObjectivePunctuation() / totalObj)  # Update Fitness
62.         return self.fitness
63.
64.     # Getters and Setters
65.     def getLarge(self):
```

```
        return self.large

    def setLarge(self, large):
        self.large = large

    def getObjectivePunctuation(self):
        return self.objectivePunctuation

    def setObjectivePunctuation(self):
        self.objectivePunctuation = self.calcObjPunc()

    def getFitness(self):
        return self.fitness

    def setFintess(self, fitness):
        self.fitness = fitness

    """
    def copy(self, another_crom, start, end):
        for i in range(start, end):
            self.route[i] = (another_crom.route[i])
    """

    def mutate(self):
        swapPos1 = random.randint(1, self.getLarge() - 1)
        swapPos2 = random.randint(1, self.getLarge() - 1)
        while swapPos1 == swapPos2:
            swapPos2 = random.randint(1, self.getLarge() -1)
        newRoute = []
        for i in range(self.getLarge()):
            if i != swapPos1 and i != swapPos2:
                newRoute.append(self.route[i])
            elif i == swapPos1:
                newRoute.append(self.route[swapPos2])
                   elif i == swapPos2:
                       newRoute.append(self.route[swapPos1])
               print("Mutated Chrom in positions:", swapPos1, "and", swapPos2, ": ", self.route)
```

population.py

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

from chromosome import Chromosome
import random


class Population(object):
    # Constructor / Instance Attributes
    def __init__(self, numChroms, cities, crossProb, mutProb):
        self.population = []  # Initial Population (Array of Chromosomes)
        self.totalObjPunc = 0  # The Sum of All Objective Functions Punctuation
        self.totalFitness = 0  # The Sum of All Objective Values
        self.numChroms = numChroms
        self.chromSize = len(cities)
        self.crossProb = crossProb
        self.mutProb = mutProb
        self.bestTrack = None
        self.bestTrackDistance = 1000000

        print("Start Algorithm")
        for i in range(numChroms):
            print("Chrom ", i, end=": ")
```

```
            # Initialization of Chromosomes
            self.population.append(Chromosome(self.chromSize, cities, None))  # Add to Population

    # Show Actual Population and Stats
    def showPopulation(self, numIter):
        self.setTotalFitness(0)
        self.setTotalObjPunc(self.calcTotalObjPunc())
        large = self.getChromSize()
        averageObjPunc = self.getTotalObjPunc() / len(self.population)
        fitness = 0
        maxVal = 0
        secondMaxVal = 0
        minVal = 0
        bestRoutePos = 0
        secondBestRoutePos = 0
        worstRoutePos = 0
        print()
        print()
        print("Population ", (numIter + 1), ":")
        for i in range(len(self.population)):
            fitness = self.population[i].calcFitness(self.getTotalObjPunc())
            self.updateTotalFitness(fitness)
            if i == 0:
                maxVal = fitness
                minVal = fitness
            elif fitness > maxVal:
                maxVal = fitness
                bestRoutePos = i
            elif fitness < minVal:
                minVal = fitness
                worstRoutePos = i
            elif (fitness > secondMaxVal) and (fitness < maxVal):
                secondMaxVal = fitness
                secondBestRoutePos = i
            # for j in range(large):
                # print(self.population[i].getRoute()[j], end=', ')
            print(self.population[i].getRoute())
        fitness = self.getTotalFitnessAverage()

        # Setting Current Generation's Best Values (If Improves Actual Best)
        if self.population[bestRoutePos].getAccumulatedDistance() < self.getBestTrackDistance():
            self.setBestTrack(self.population[bestRoutePos])
            self.setBestTrackDistance(self.population[bestRoutePos].getAccumulatedDistance())

        print()
        print("Generation", numIter+1, "Final Status:")
        print("Worst Value: Route Nº", worstRoutePos, "with:",
              self.population[worstRoutePos].getObjectivePunctuation(), "OP,", round(minVal, 4), "Fit
")
        print("Best Value: Route Nº", bestRoutePos, "with:",
              self.population[bestRoutePos].getObjectivePunctuation(), "OP,", round(maxVal, 4), "Fit"
)
        print("Total Distance of Best Route:", self.getBestTrackDistance(), "km")
        print("Average OP:", averageObjPunc, "--
- Average Fitness:", round(self.getTotalFitnessAverage(), 6))
        print("---")
        print()

        elitChrom = self.population[bestRoutePos]
        secondElitChrom = self.population[secondBestRoutePos]

        return {
            "AverageOP": averageObjPunc,
            "MinOP": self.population[worstRoutePos].getObjectivePunctuation(),
```

```
            "MaxOP": self.population[bestRoutePos].getObjectivePunctuation(),
            "ElitChrom": elitChrom,
            "SecondElitChrom": secondElitChrom
        }

    # Get the Best Values
    def getBestTrackAg(self):
        return self.getBestTrack(), self.getBestTrackDistance()

    # Calculate Total of Objective Functions Punctuation in the actual Generation
    def calcTotalObjPunc(self):
        acumObjPunc = 0
        for chromosome in self.population:
            acumObjPunc += chromosome.getObjectivePunctuation()  # Add Every Objective Function Punct
uation
        return acumObjPunc

           # Update Total Fitness
           def updateTotalFitness(self, fitness):
               self.totalFitness += fitness

           def getBestTrack(self):
               return self.bestTrack

           def setBestTrack(self, track):
               self.bestTrack = track

           def getBestTrackDistance(self):
               return self.bestTrackDistance

           def setBestTrackDistance(self, trackDistance):
               self.bestTrackDistance = trackDistance

           # Add to Population
           def addChrom(self, Chrom):
               self.population.append(Chrom)

           # Reproduction
           def reproduce(self, elitChrom, secondElitChrom):
               parents = []  # List of Potential Parents
               newGeneration = []  # List of Children

               # Elitism
               parents.append(elitChrom)
               parents.append(secondElitChrom)

               print("Roulette Results: ", end='')
               for _ in range(len(self.population)):
                   indexSelectedParent = self.roulette()  # Parents Selected by Roulette
                   if indexSelectedParent is not None:
                       parents.append(self.population[indexSelectedParent])
                   else:
                       parents.append(self.population[0])  # Elite Chromosome
                       print(0, end=', ')
               print()
               for i in range(2, len(parents), 2):
                   father1 = parents[i]
                   father2 = parents[i + 1]
                   if self.crossPosibility():  # CrossOver Probability Evaluation
                       son1, son2 = self.cross(father1, father2)  # CrossOver
                       print("Successful CrossOver in reproduction:", (i + 2) / 2)  # Only Print
                   else:
                       son1, son2 = self.copy(father1, father2)  # Direct Assignation (Without CrossO
ver)
```

```
                    print("CrossOver didn't happen in reproduction:", (i + 2) / 2)  # Only Print
                # Individual Mutation Probability Evaluation
                if self.mutationPosibility():
                    son1.mutate()
                if self.mutationPosibility():
                    son2.mutate()

                son1.setObjectivePunctuation()
                son2.setObjectivePunctuation()

                self.addChildren(son1, son2, newGeneration)
            self.replacePopulation(newGeneration)
            self.setTotalFitness(0)

        # Genetic Operator (Roulette Method)
        def roulette(self):
            # Generator of a Bidimensional List (Fitness Range of Chromosomes)
            newRoulette = [[0] * 2 for _ in range(len(self.population))]
            acum = 0  # Acumulator of Relative Fitness from 0 to 1 (Fills Roulette)
            for i in range(len(self.population)):
                newRoulette[i][0] = acum  # Range Min: Last Acum Value
                acum += self.population[i].getFitness()  # Acum's Value From Zero
                newRoulette[i][1] = acum  # Range Max: New Acum Value
            ranNum = random.uniform(0, 1)  # Random Number from 0.000000 to 0.999999
            for i in range(len(newRoulette)):
                if newRoulette[i][0] <= ranNum <= newRoulette[i][1]:
                    # Return Selected Chromosome if the Random Number Exists in its Range
                    print(i, end=', ')
                    return i
            return None  # Error Return

        def crossPosibility(self):  # CrossOver posibility evaluation
            if self.getCrossProb()*100 >= random.randint(1, 100):
                return True
            else:
                return False

        # Cyclic Crossover
        def cross(self, parent1, parent2):
            crom_size = parent1.getLarge()

            newRoute1 = ['']*crom_size
            newRoute2 = ['']*crom_size

            newRoute1[0] = parent1.route[0]
            aux = parent2.route[0]
            pos = None
            while aux != parent1.route[0]:
                for i in range(crom_size):
                    if parent1.route[i] == aux:
                        pos = i
                        break
                newRoute1[pos] = aux
                aux = parent2.route[pos]

            for i in range(crom_size):
                if newRoute1[i] == '':
                    newRoute1[i] = parent2.route[i]

            newRoute2[0] = parent2.route[0]
            aux = parent1.route[0]
            pos = None
            while aux != parent2.route[0]:
                for i in range(crom_size):
```

```
                    if parent2.route[i] == aux:
                        pos = i
                        break
                newRoute2[pos] = aux
                aux = parent1.route[pos]

            for i in range(crom_size):
                if newRoute2[i] == '':
                    newRoute2[i] = parent1.route[i]

            son1 = Chromosome(crom_size, None, newRoute1)
            son2 = Chromosome(crom_size, None, newRoute2)

            print()
            print("New Sons after CrossOver: ")
            print(son1.getRoute())
            print(son2.getRoute())

            return son1, son2

        def copy(self, chrom1, chrom2):
            crom_size = chrom1.getLarge()
            newRoute1 = chrom1.getRoute()
            newRoute2 = chrom2.getRoute()
            son1 = Chromosome(crom_size, None, newRoute1)
            son2 = Chromosome(crom_size, None, newRoute2)

            print()
            print("New Sons without CrossOver (Indentical): ")
            print(son1.getRoute())
            print(son2.getRoute())

            return son1, son2

        def mutationPosibility(self):  # Mutation posibility evaluation
            if self.getMutProb() * 100 >= random.randint(1, 100):
                return True
            else:
                return False

        def addChildren(self, son1, son2, newGeneration):
            newGeneration.append(son1)
            newGeneration.append(son2)

        def replacePopulation(self, newGeneration):  # Replace All Population in every Iteration
            self.population = []
            for i in range(len(newGeneration)):
                self.population.append(newGeneration[i])

        def getTotalObjPunc(self):
            return self.totalObjPunc

        def setTotalObjPunc(self, total):
            self.totalObjPunc = total

        def getTotalFitness(self):
            return self.totalFitness

        def setTotalFitness(self, total):
            self.totalFitness = total

        def getTotalFitnessAverage(self):
            return self.totalFitness / len(self.population)

```

```
275.            def getNumChroms(self):
276.                return self.numChroms
277.
278.            def setNumChroms(self, numChroms):
279.                self.numChroms = numChroms
280.
281.            def getChromSize(self):
282.                return self.chromSize
283.
284.            def setChromSize(self, chromSize):
285.                self.chromSize = chromSize
286.
287.            def getCrossProb(self):
288.                return self.crossProb
289.
290.            def setCrossProb(self, crossProb):
291.                self.crossProb = crossProb
292.
293.            def getMutProb(self):
294.                return self.mutProb
295.
296.            def setMutProb(self, mutProb):
297.                self.mutProb = mutProb
```

FRONTEND

index.html

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.      <head>
4.          <meta charset="UTF-8">
5.          <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.          <meta http-equiv="X-UA-Compatible" content="ie=edge">
7.          <link rel="stylesheet" type="text/css" href="../static/css/styles.css">
8.          <title>Traveling Salesman Problem</title>
9.      </head>
10.     <body style="background-image:url(../static/resources/artificial-
    intelligence.jpg)" onload='initialization()'>
11.         <header>
12.         <!-- <body style="background-image:url({{url_for('static', filename='artificial-
    intelligence.jpg')}})"> -->
13.             <div class="title"><h1>Genetic Algorithms TP3: Traveling Salesman Problem</h1></div>
14.         </header>
15.         <main>
16.             <ul id="options">
17.                 <li><a href="#" id="opt1" class="menuLink">
18.                     EXERCISE 2-A
19.                     <br/>
20.                     Best Selected City's Route by Heuristic
21.                 </a></li>
22.                 <li><a href="http://127.0.0.1:5000/Map" id="opt2" class="menuLink">
23.                     EXERCISE 2-B
24.                     <br/>
25.                     Best All Cities' Route by Heuristic
26.                 </a></li>
27.                 <li><a href="http://127.0.0.1:5000/Map/Ag" id="opt3" class="menuLink">
28.                     EXERCISE 2-C
29.                     <br/>
30.                     Best All Cities' Route by Genetic Algorithm
31.                 </a></li>
```

```
            </ul>
        </main>
        <footer>
            <p>MADE BY: Antonelli, Recalde, Rohn</p>
            <br/>
            <p>Genetics Algorithms 2019 - UTN FRRo</p>
        </footer>
        <div id="overlayModal" class="modal">
            <div class="modal-content">
                <div class="modal-header">
                    <span class="close">×</span>
                    <h2>Select One City</h2>
                </div>
                <div class="modal-body">
                    <input type="text" id="cityName" name="cityName" placeholder="City Name">
                    <button id="citySend">Run!</button>
                </div>
            </div>
        </div>
        <div id="agModal" class="modal">
            <div class="modal-content">
                <div class="modal-header">
                    <h2>Genetic Algorithm</h2>
                </div>
                <div class="modal-body">
                    <p>Training Genetic Algorithm Model, please wait...</p>
                </div>
            </div>
        </div>
        <script src="../static/js/functions.js" type="text/javascript"></script>
    </body>
</html>
```

showMap.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>TSP: Map</title>
        <link rel="stylesheet" href="../static/css/styles.css">
        <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBGsPRU6-v05wBr_dfzUj-
AkQn3HvpCezk&callback=initMap"
        defer></script>
        <script src="../static/js/mapConfiguration.js" type="text/javascript"></script>
        <script type="text/javascript">
            document.addEventListener('DOMContentLoaded', function () {
                setFlightPath({{ track|tojson }});
                setMarkers({{ track|tojson }});
            });
        </script>
    </head>
    <body onload='logTrack({{ track|tojson }})'>
        <div id="map-container">
            <div id="map"></div>
            <div id="sideStats"></div>
        </div>
        <script src="../static/js/functions.js" type="text/javascript"></script>
    </body>
</html>
```

styles.css

```css
/* ############ PRINCIPAL STYLES ############ */
* {
    margin: 0;
    padding: 0;
}

body {
    background-color: darkred;
    /* background-image: url("route/img"); */
    background-repeat: no-repeat;
    background-size: cover;
}

body, html{
    height: 100vh;
}

header { height: 15vh; }

main { height: 75vh; }

footer {
    height: 10vh;
    color: white;
    background-color: black;
    padding: 1rem;
    text-align: center;
}

.title {
    margin: 2rem;
    display: flex;
    flex-direction: row;
    justify-content: center;
    color: white;
    text-shadow: -2px 0 black, 0 2px black, 2px 0 black, 0 -2px black;
}

.title>h1 {
    padding: 1rem;
    background-color: saddlebrown;
    border: 0.25rem dashed cornsilk;
}

#options {
    list-style: none;
    display: flex;
    flex-direction: column;
    text-align: center;
    align-items: center;
    /* justify-content: center; */
}

#options>li {
    margin: 1rem;
    font-size: 150%;
    display: flex;
}

#options>li>a {
    padding: 1.5rem 15vw 1.5rem 15vw;
```

```
    border: 0.25rem dashed cornsilk;
    background-color: slateblue;
}

/* All Links */
a { text-decoration: none; }

/* Unvisited Links */
.menuLink:link { color: black; }
.whiteLink:link { color: white; }

/* Visited Links */
.menuLink:visited { color: black; }
.whiteLink:visited { color: white; }

/* MouseOver Links */
.menuLink:hover { color: cornflowerblue; }
.whiteLink:hover { color: cornflowerblue; }

/* Selected Links */
.menuLink:active { color: cornflowerblue; }
.whiteLink:active { color: cornflowerblue; }

/* ############# SHOWMAP STYLES ############# */

#map-container {
    display: flex;
    flex-direction: row;
    width: 100vw;
    height: 100vh;
}

#map {
    /* flex: 1 1 auto; */
    width: 65vw;
    padding-bottom: 10px;
}

      #sideStats {
          /* flex: 1 1 auto; */
          color: white;
          background-color: black;
          width: 35vw;
          padding-top: 10px;
          padding-bottom: 10px;
          text-align: center;
          overflow: scroll;
      }

      /* ############# MODAL STYLES ############# */

      /* Modal Background */
      .modal {
        display: none; /* Hidden by default */
        position: fixed;
        z-index: 10;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        overflow: auto;
        background-color: rgb(0,0,0);
        background-color: rgba(0,0,0,0.4); /* Black w/opacity */
      }
```

```
126.
127.       /* Modal Box */
128.       .modal-content {
129.         background-color: #FEFEFE;
130.         margin: 15% auto;
131.         padding: 0;
132.         border: 1px solid #888;
133.         width: 50%;
134.       }
135.
136.       /* Modal Header */
137.       .modal-header {
138.         padding: 2px 16px;
139.         background-color: darkslategrey;
140.         color: white;
141.       }
142.
143.       /* Modal Body */
144.       .modal-body { padding: 20px 16px; }
145.
146.       /* Modal Footer */
147.       /*.modal-footer {
148.         padding: 2px 16px;
149.         background-color: #5cb85c;
150.         color: white;
151.       }*/
152.
153.       /* Closing Styles */
154.       .close {
155.         color: #aaa;
156.         float: right;
157.         font-size: 28px;
158.         font-weight: bold;
159.       }
160.
161.       .close:hover, .close:focus {
162.         color: black;
163.         text-decoration: none;
164.         cursor: pointer;
165.       }
```

functions.js

```
1.  // Stats of the Track (Map Events)
2.  function logTrack(track) {
3.      console.log("Stats Loading...");
4.      document.getElementById("sideStats").innerHTML = "<p>" + "BEST ROUTE STATS:" + "</p>" + "<br />";

5.      document.getElementById("sideStats").innerHTML += "<p style='padding-bottom: 10px'>" +
6.          "Total Distance: " +  track["accumulated_distance"] + " km" + "</p>" + "<hr />" + "<br />";
7.      track["cities_to_visit"].forEach(function (item) {
8.          document.getElementById("sideStats").innerHTML += "<p>" + item.name + ": " +
9.          item.lat + " | " + item.lng + "</p>" + "<br />";
10.         console.log(item.name, item.lat, item.lng);
11.     });
12.     document.getElementById("sideStats").innerHTML += "<hr />";
13.     document.getElementById("sideStats").innerHTML += "<div id='goBack' style='padding-
    top: 10px'>" +
14.         "<a href='#' class='whiteLink'>" + "\u21a9 Back to Index" + "</a>" + "</div>" + "<br />";
15.     console.log("Total Distance: " + track["accumulated_distance"] + " km");
16.
17.     // "Go Back" Button
18.     let backBtn = document.getElementById("goBack");
```

```
19.
20.     // URL Root
21.     let urlRoot = "http://127.0.0.1:5000/";
22.
23.     // Back to Index (Doesn't Work, Bubbling problem!)
24.     backBtn.addEventListener('click', function (e) {
25.         // backBtn.onclick = function() { window.location.href = urlRoot; };
26.        e.preventDefault();
27.        window.location.href = urlRoot;
28.     }, true);
29.
30.     console.log("Fully Loaded!");
31. }
32.
33. // Index Events
34. function initialization() {
35.     // Get the Modals
36.     let overlayModal = document.getElementById("overlayModal");
37.     let agModal = document.getElementById("agModal");
38.
39.     // Get the Button that opens the Modal in both Modals
40.     let btn = document.getElementById("opt1");
41.     let btnAg = document.getElementById("opt3");
42.
43.     // Get the <span> element that closes the Modal in OverlayModal
44.     let closeSpan = document.getElementsByClassName("close")[0];
45.
46.     // When the user clicks on the Button, open the Modal
47.     btn.onclick = function() {
48.       overlayModal.style.display = "block";
49.     };
50.
51.     btnAg.onclick = function() {
52.       agModal.style.display = "block";
53.     };
54.
55.     // When the user clicks on <span> (x), close the Modal
56.     closeSpan.onclick = function() {
57.       overlayModal.style.display = "none";
58.     };
59.
60.     // When the user clicks anywhere outside of the Modal, close it
61.     window.onclick = function(event) {
62.       if (event.target === overlayModal) {
63.         overlayModal.style.display = "none";
64.       }
65.     };
66.
67.     // Root URL
68.     let urlRoot = "http://127.0.0.1:5000/";
69.
70.     // URI with Map Endpoint
71.     let uriMap = urlRoot + "Map/";
72.
73.     // Run! inside-modal button
74.     let runBtn = document.getElementById("citySend");
75.
76.     // Reload Page (Exercise 2-A)
77.     runBtn.onclick = function() {
78.         // City name
79.         let cityName = document.getElementById("cityName").value;
80.         window.location.href = uriMap + cityName;
81.     };
82. }
```

configurations.js

```
function initMap() {
    // Central reference location, UTN FRRo
    baseLocation = {lat: -41.0000000, lng: -64.0000000};
    // Map, hiding business and other unnecessary stuff
    map = new google.maps.Map(
        document.getElementById('map'), {
            center: baseLocation
            ,zoom: 4.25
            ,styles: [{"featureType": "poi.business","stylers": [{ "visibility": "off" }]}]
            ,mapTypeControl: false
            ,fullscreenControl: false
            ,streetViewControl: false
        });

    locationsToAdd = [];

    // To draw the track
    // https://stackoverflow.com/questions/31305497/how-to-draw-an-arrow-on-every-polyline-segment-on-google-maps-v3
    flightPath = new google.maps.Polyline({
       path: locationsToAdd
       ,geodesic: true
       ,strokeColor: '#FF000'
       ,strokeOpacity: 0.6
       ,strokeWeight: 4
       ,icons: [{
           icon: {path: google.maps.SymbolPath.FORWARD_CLOSED_ARROW}
           ,offset: '100%'
           ,repeat: '200px'
        }]
    });

    flightPath.setMap(map);

    // Center the map taking into account every position
    bounds = new google.maps.LatLngBounds();

    markers = [];
}

function setFlightPath(track) {
    track["cities_to_visit"].forEach(function (location) {
        let locationToAdd = {
            lat: parseFloat(location.lat)
            ,lng: parseFloat(location.lng)
        };

        console.log(locationToAdd);

        locationsToAdd.push(locationToAdd);
    });

    locationsToAdd.forEach(function (location) {
        console.log(location.lat, location.lng);
    });

    flightPath.setPath(locationsToAdd);

    console.log(flightPath);
}

```

```
function setMarkers(track) {
    let locations = track["cities_to_visit"];

    let location = locations[locations.length - 1];

    console.log(location);

    let locationToAdd = {
        lat: parseFloat(location.lat)
        ,lng: parseFloat(location.lng)
    };

    newMarker(locationToAdd);
}

function newMarker(locationToAdd) {
    let marker = new google.maps.Marker({
       position: locationToAdd
       ,map: map
    });

    markers.push(marker);
}
```

LENGUAJES UTILIZADOS Y OTRAS TECNOLOGÍAS:

- ✓ ***BACKEND*: Python, Flask (Framework), API Google Maps**
- ✓ ***FRONTEND*: HTML5, CSS3, JavaScript**

REPOSITORIO GITHUB:

https://github.com/utn-frro-geneticos19-g16/

Explicacion de Funcionamiento

Ejercicio 1: Búsqueda Exhaustiva

Podemos decir que la búsqueda de la ruta de distancia mínima que logra unir a todas las capitales de la republica argentina utilizando el método exhaustivo es prácticamente imposible ya que llevaría miles de años su resolución por ser el espacio de soluciones tan grande (24^24).

Ejercicio 2: Método Heurístico, e implementación de Algoritmo Genético

Para resolver los ejercicios 2a y 2b, implementamos la heurística de "Visitar la ciudad siguiente no-visitada más cercana a la ciudad actual", y luego volver al inicio. En el inciso 2a se pide que dada una ciudad, se obtenga el mejor recorrido posible siguiendo esta heurística, y en el 2b: realizar lo anterior a cada provincia, y quedarse con el mejor recorrido posible, que en este caso llegamos a la conclusión de que es: partir desde la ciudad de **Neuquén**.

Más abajo explicaremos donde se encuentra y que hacen los métodos que componen los archivos de nuestra solución, pero en síntesis tenemos objetos ciudades donde cada una tiene un atributo que es un diccionario con la distancia respecto a todas las otras ciudades. Lo que hacemos es crear una lista de "Ciudades a Visitar" que son también objetos, pero solo el nombre de la ciudad, su latitud y su longitud real (para utilizar en el Mapa de Google Maps luego). Vamos en cada momento agregando la ciudad actual a esta lista, y preguntando cuál es la siguiente más cercana (corroborando no repetir las que ya están en la lista). Por su puesto, acumulando la distancia parcial entre ciudad y ciudad. En el final, le enviamos la lista de ciudades a visitar con latitudes y longitudes, y la distancia total recorrida, a nuestro mapa para que grafique el recorrido.

Para el Algoritmo Genético que se pide en el 2c, lo que le enviamos al mapa es lo mismo: la mejor ruta obtenida con nombres para exhibir a un lado y coordenadas para graficar el recorrido, y la distancia total. Los parámetros iniciales (tamaño de población, iteraciones, probabilidad de mutación y crossover) se encuentran en "citiesManager.py", y todos los elementos que componen al algoritmo genético los encontraremos en "population.py" y "chromosome.py" (mostrar población actual, cálculo de función objetivo, cálculo de Fitness, reproducción, ruleta, crossover, mutación, etc…). Abajo, hay más detalles sobre estos métodos.

Explicación del Funcionamiento de los métodos y archivos de ejercicios 2a, 2b, 2c

app.py

- Aquí se da la inicialización de nuestra aplicación principal junto con las ciudades.
- También tenemos la lógica del web server donde cada URL se le asigna una funcionalidad con una entrada del Query String y un Template renderizado.

city.py

- Contiene la clase **City** con sus respectivas funciones que devuelven los datos de la clase.
- La función **get_nearest_neig** recibe un listado de las ciudades actualmente visitadas y compara con el diccionario de ciudades vecinas para encontrar la capital más próxima descartando las que ya han sido visitadas.
- La función **get_distance_to** devuelve la distancia de la ciudad actual a otra.

cityToVisit.py

- Contiene a la clase **CityToVisit** donde tenemos solamente la información necesaria para mostrar.

neighbourCitiesRepo.py

- Contiene a la clase llamada **NeighbourCitiesRepo** la cual básicamente es un repositorio donde se encuentran definidas, en un diccionario, todas las capitales donde cada una contiene los nombres y distancias respectivas a todas las otras capitales de la República Argentina.
- Esta clase además tiene la función **get_near_cities_dict** que al ser llamada devuelve la ciudad indicada.

citiesManager.py

- Contiene la clase **CitiesManager** donde se inicializan todas las ciudades instanciando cada una con el objeto City y agregándolas a un nuevo diccionario.
- La función **get_best_track_from** busca y devuelve el mejor recorrido que une a todas las capitales a partir de una ciudad dada, para ello inicia el recorrido con la capital raíz dada y a medida que avanza hacia las ciudades vecinas va acumulando la distancia total recorrida además de ir agregando a un array cada ciudad visitada.
- La función **get_best_track** busca y devuelve el mínimo recorrido entre todas las capitales.
- La función **get_track_with_ag** busca y devuelve el mejor recorrido que une a todas las capitales utilizando algoritmos genéticos.

chromosome.py

- Mostrar sus genes en diferentes formatos (cada gen es una ciudad)
- Contiene la clase Chromosome donde cada cromosoma es un track (recorrido)
- Calcular función objetivo a partir de hacer el inverso a la sumatoria de todas las distancias entre ciudades a medida que avanza el recorrido
- Calcular su Fitness a partir de dividir la valuación de la función objetivo por la valuación de la función objetivo acumulada por todos los cromosomas

population.py

- Mostrar los cromosomas que la conforman
- Inicializarse con el número de cromosomas, la probabilidad de crossover y la probabilidad de mutación indicados
- Reproducirse, lo que incluiría la selección y crossover
- Seleccionar a los cromosomas dentro de una ruleta de manera probabilística según su Fitness
- Crear una ruleta que permita a los cromosomas más aptos tener mayores posibilidades de ser seleccionados, pero utilizando un rango de números reales para nuestra ruleta (Fitness Acumulativo)
- Calcular la puntuación objetivo acumulada por todos los cromosomas que la conforman
- Calcular el mínimo, máximo y promedio en cada generación (al inicializarse y después de cada reproducción)
- El crossover implementado es de tipo cíclico, pues se mantiene la estructura de genes todos diferentes intercambiando posiciones pero no se duplica ninguno, cosa indispensable en el TSP.
- La mutación que seleccionamos fue Swap Mutation, que en definitiva cambia N genes (en nuestro caso N=1) de posición entre sí, y tampoco hay repetición de cromosomas. Otra alternativa a Swap Mutation que también sirve para el TSP, entre otras, pudo haber sido Shift Mutation.

- Aplicación de Elitismo, cuya idea es hacer que los mejores cromosomas de una generación se mantengan en la generación siguiente. Nosotros aplicamos elitismo siempre a los 2 mejores cromosomas de la generación actual
- **Para aplicar el Elitismo hicimos lo siguiente**: calculamos las funciones objetivo de todos los cromosomas de la generación actual, y guardamos cuáles son los 2 con puntajes más altos; luego cuando llamamos al método que comienza con el proceso de selección y reproducción (lo primero que sucede en este método es instancias 2 listas vacías, una para los padres candidatos, y otra que tendrá los hijos de cada reproducción que se van a ir agregando a ésta para en el final reemplazar la población actual), y les pasamos los 2 cromosomas élites directamente a la lista de hijos (llamada NewGeneration) de forma que no haya manera de que se pierdan. Luego, como cualquier otro cromosoma, participarán de forma independiente de la ruleta, crossover y mutación. La siguiente generación tendrá entonces en su población a ambos cromosomas élites.

Archivos HTML5 y CSS3 (Frontend)

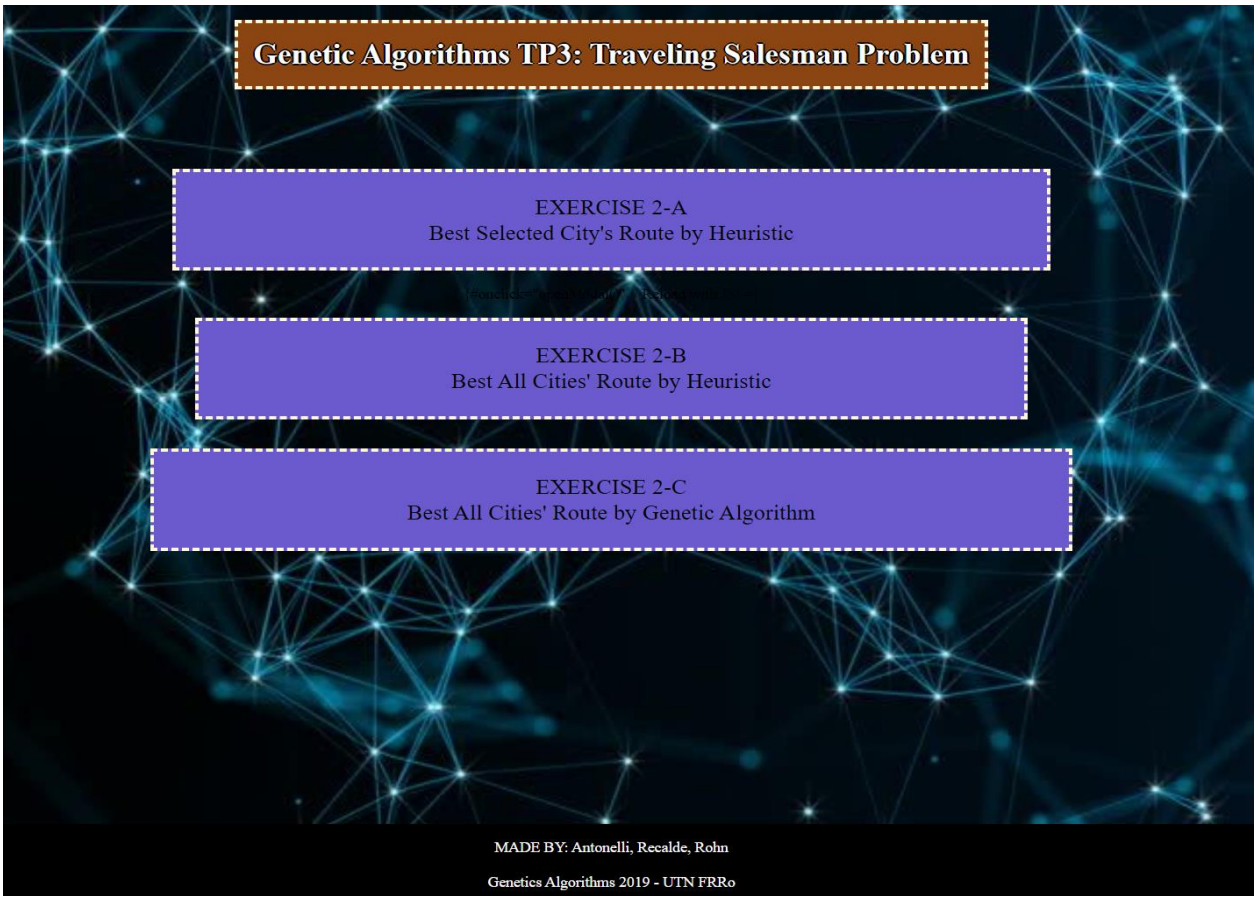
- Para enlazar Python con HTML y CSS, utilizamos un Framework llamado Flask
- Pantalla inicial index.html con la posibilidad de seleccionar el ejercicio correspondiente (Heurísticas y Algoritmo Genético)
- Pantalla showMap.html con el recorrido a exhibir en el mapa de Google Maps y la cantidad de kilómetros total, según lo que el resultado del ejercicio seleccionado le envíe
- Todos los estilos que utilizamos en styles.css.

Archivos JavaScript (Frontend)

- Tenemos unos archivos: functions.js y mapConfigurations.js que es solo para comunicar correctamente la parte visual de nuestra página con la API de Google Maps, y exhibir lo que pide los ejercicios (lista de las ciudades recorridas y distancia total) más botón para volver al inicio y detalles.
- Solo lo utilizamos para eso, pues toda la lógica y funcionalidad de los ejercicios los hicimos totalmente en lenguaje Python.

Salida de Pantalla

Menú Principal:



Ejercicio 2-A

BEST ROUTE STATS:

Total Distance: 9781 km

Cordoba: -31.4134998 | -64.1810532

San Luis: -33.2950096 | -66.3356323

Mendoza: -32.8908386 | -68.8271713

San Juan: -31.5375004 | -68.5363922

La Rioja: -29.4110508 | -66.8506699

S.F.d.V.d. Catamarca: -28.4695702 | -65.7852402

Sgo. del Estero: -27.7951107 | -64.2614899

S.M. de Tucuman: -26.8241405 | -65.2226028

Salta: -24.7859001 | -65.4116592

S.S. de Jujuy: -24.1945705 | -65.2971191

Resistencia: -27.4605598 | -58.9838905

Corrientes: -27.4806004 | -58.8340988

Formosa: -26.1775303 | -58.1781387

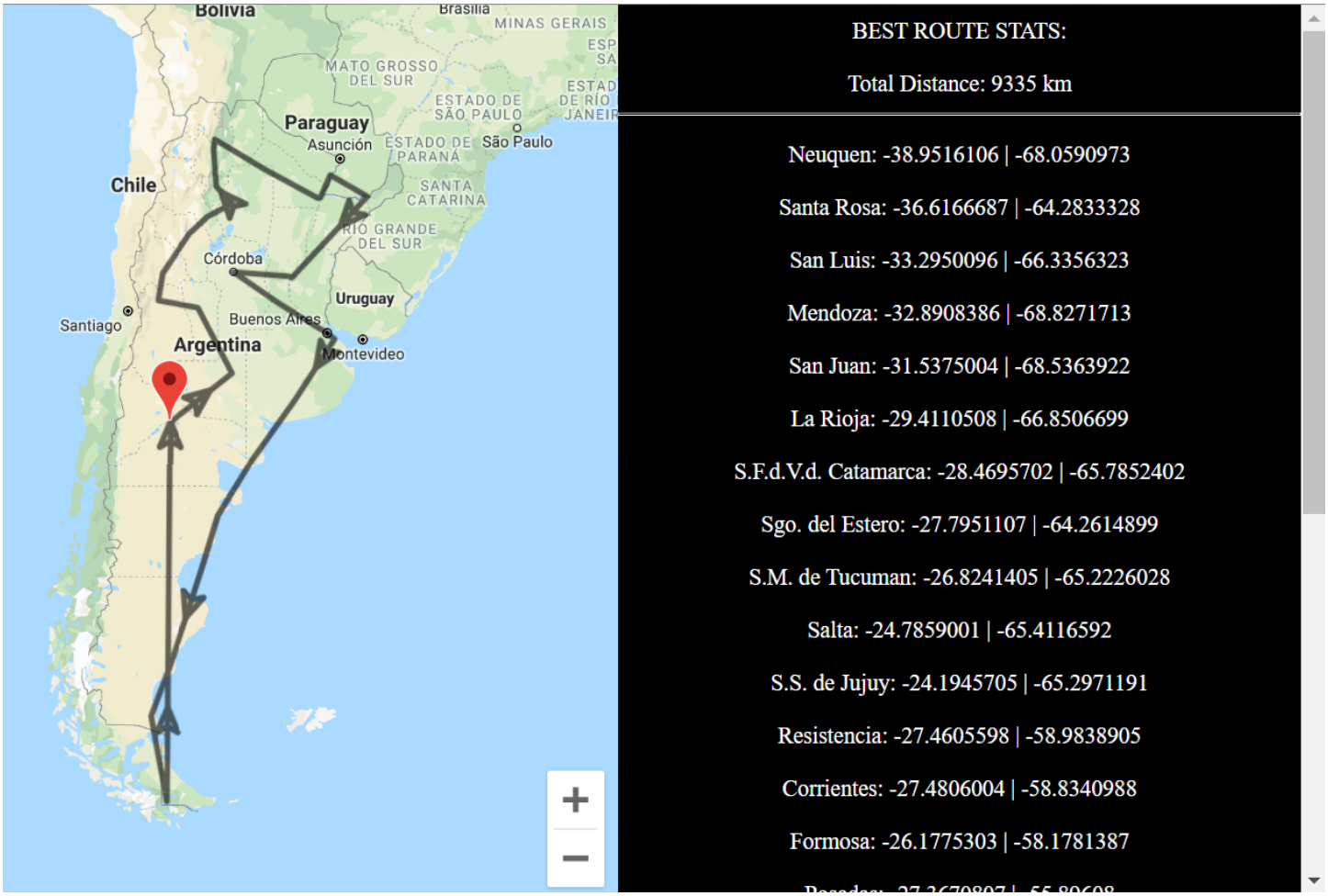
Posadas: -27.3670807 | -55.89608

Parana: -31.7319698 | -60.5237999

Santa Fe: -31.6333294 | -60.7000008

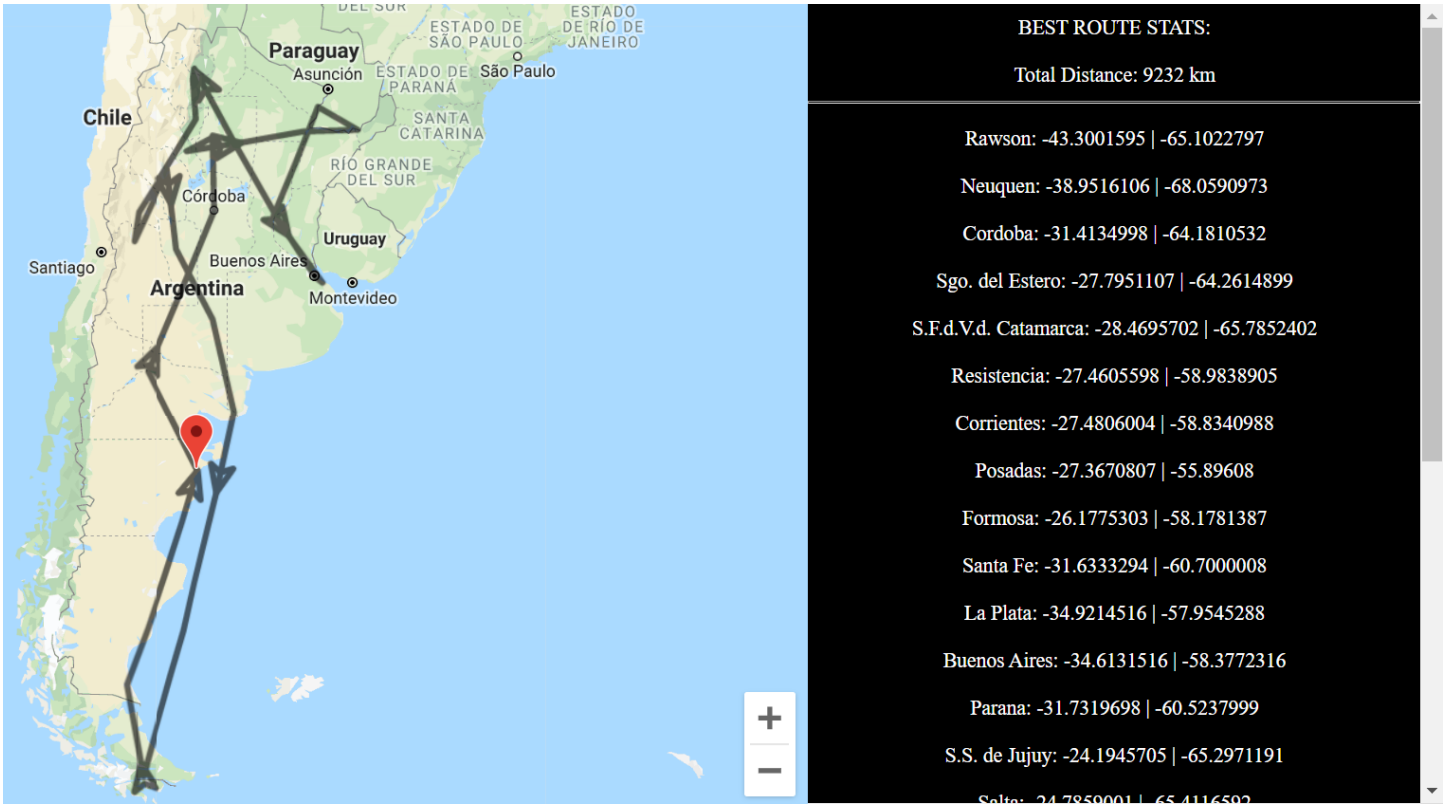
Cordoba: -31.4134998 | -64.1810532 , San Luis: -33.2950096 | -66.3356323 , Mendoza: -32.8908386 | -68.8271713 , San Juan: -31.5375004 | -68.5363922 , La Rioja: -29.4110508 | -66.8506699 , S.F.d.V.d. Catamarca: -28.4695702 | -65.7852402 , Sgo. del Estero: -27.7951107 | -64.2614899 , S.M. de Tucuman: -26.8241405 | -65.2226028 , Salta: -24.7859001 | -65.4116592 , S.S. de Jujuy: -24.1945705 | -65.2971191 , Resistencia: -27.4605598 | -58.9838905 , Corrientes: -27.4806004 | -58.8340988 , Formosa: -26.1775303 | -58.1781387 , Posadas: -27.3670807 | -55.89608 , Parana: -31.7319698 | -60.5237999 , Santa Fe: -31.6333294 | -60.7000008 , Buenos Aires: -34.6131516 | -58.3772316 , La Plata: -34.9214516 | -57.9545288 , Santa Rosa: -36.6166687 | -64.2833328 , Neuquen: -38.9516106 | -68.0590973 , Viedma: -40.8134499 | -62.9966812 , Rawson: -43.3001595 | -65.1022797 , Rio Gallegos: -51.6226082 | -69.218132 , Ushuaia: -54.7999992 | -68.3000031 , Cordoba: -31.4134998 | -64.1810532 , Total Distance = 9781

Ejercicio 2-B



Neuquen: -38.9516106 | -68.0590973 , Santa Rosa: -36.6166687 | -64.2833328 , San Luis: -33.2950096 | -66.3356323 , Mendoza: -32.8908386 | -68.8271713 , San Juan: -31.5375004 | -68.5363922 , La Rioja: -29.4110508 | -66.8506699 , S.F.d.V.d. Catamarca: -28.4695702 | -65.7852402 , Sgo. del Estero: -27.7951107 | -64.2614899 , S.M. de Tucuman: -26.8241405 | -65.2226028 , Salta: -24.7859001 | -65.4116592 , S.S. de Jujuy: -24.1945705 | -65.2971191 , Resistencia: -27.4605598 | -58.9838905 , Corrientes: -27.4806004 | -58.8340988 , Formosa: -26.1775303 | -58.1781387 , Posadas: -27.3670807 | -55.89608 , Parana: -31.7319698 | -60.5237999 , Santa Fe: -31.6333294 | -60.7000008 , Cordoba: -31.4134998 | -64.1810532 , Buenos Aires: -34.6131516 | -58.3772316 , La Plata: -34.9214516 | -57.9545288 , Viedma: -40.8134499 | -62.9966812 , Rawson: -43.3001595 | -65.1022797 , Rio Gallegos: -51.6226082 | -69.218132 , Ushuaia: -54.7999992 | -68.3000031 , Neuquen: -38.9516106 | -68.0590973 , Total Distance = 9335

Ejercicio 3



Last Generation Reached Correctly

Best Track: Rawson, Neuquen, Cordoba, Sgo. del Estero, S.F.d.V.d. Catamarca, Resistencia, Corrientes, Posadas, Formosa, Santa Fe, La Plata, Buenos Aires, Parana, S.S. de Jujuy, Salta, S.M. de Tucuman, San Juan, Mendoza, La Rioja, San Luis, Santa Rosa, Viedma, Ushuaia, Rio Gallegos, Rawson (Vuelta al Inicio)

Total Distance: 9232 km, after 500 generations with 300 Chromosomes

Conclusiones

Como vimos en el primer ejercicio la búsqueda exhaustiva no brindaría resultados en un intervalo de tiempo práctico.

Utilizando la búsqueda heurística obtuvimos, en muy poco tiempo, un muy buen resultado al recorrer todas las capitales yendo siempre a la más cercana.

Respecto a la implementación de algoritmos genéticos nosotros utilizamos Crossover Cíclico y Swap Mutation para mantener la estructura de los cromosomas lo cual nos evita que se corrompan y surjan ciudades repetidas dentro de los recorridos.

En un principio habíamos llegado a la conclusión de que, empíricamente, para obtener buenos resultados (distancias de entre 9200km y 10000km) debimos definir una población de al menos 200 cromosomas y al menos 1500 generaciones. Sin embargo, logramos mejorar esto bastante. Utilizamos un valor de probabilidad de mutación: de 0.1 y de crossover: de 0.75.

Al agregar elitismo notamos que la convergencia sucede muy tempranamente por lo que aumentamos la mutación a por lo menos un 0.4 para salir del estancamiento producido. Aun así debemos correr el programa varias veces para obtener un resultado aproximado o mejor al obtenido por la heurística. Hasta acá nuestro mejor recorrido rondaba cerca de 9150km. A razón de coste/beneficio de aumentar las iteraciones y/o el tamaño de la población, es una solución muy buena.

Sin embargo, experimentando un poco más, determinamos que aumentar la cantidad de cromosomas a 1000, y compensar parte del tiempo de procesamiento mayor requerido bajando la cantidad de generaciones a 200, y pasar de valores para la probabilidad de crossover: de 0.75 a 0.9 y de mutación: de 0.4 a 0.65, se obtenían resultados mucho mejores, y que un recorrido no sea bueno se volvió menos frecuente. Logramos obtener algunos recorridos de menos de 9000km.