Proyecto del vendedor ambulante

Nombre: *Addison A. Reyes C.*

Matricula: *2021-2026*

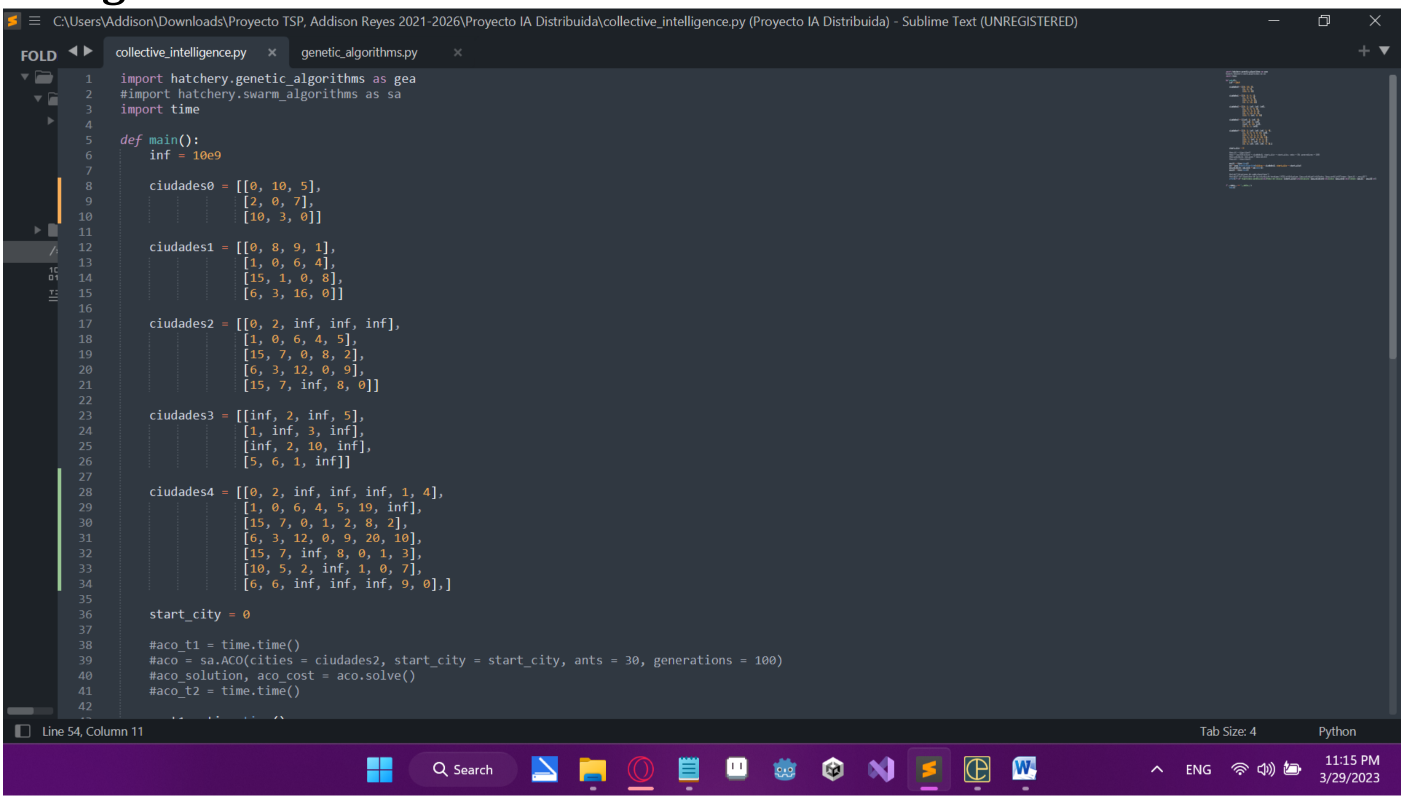
Profesor: *Carlos B. Ogando M.*

Asignatura: *Inteligencia Artificial Distribuida*

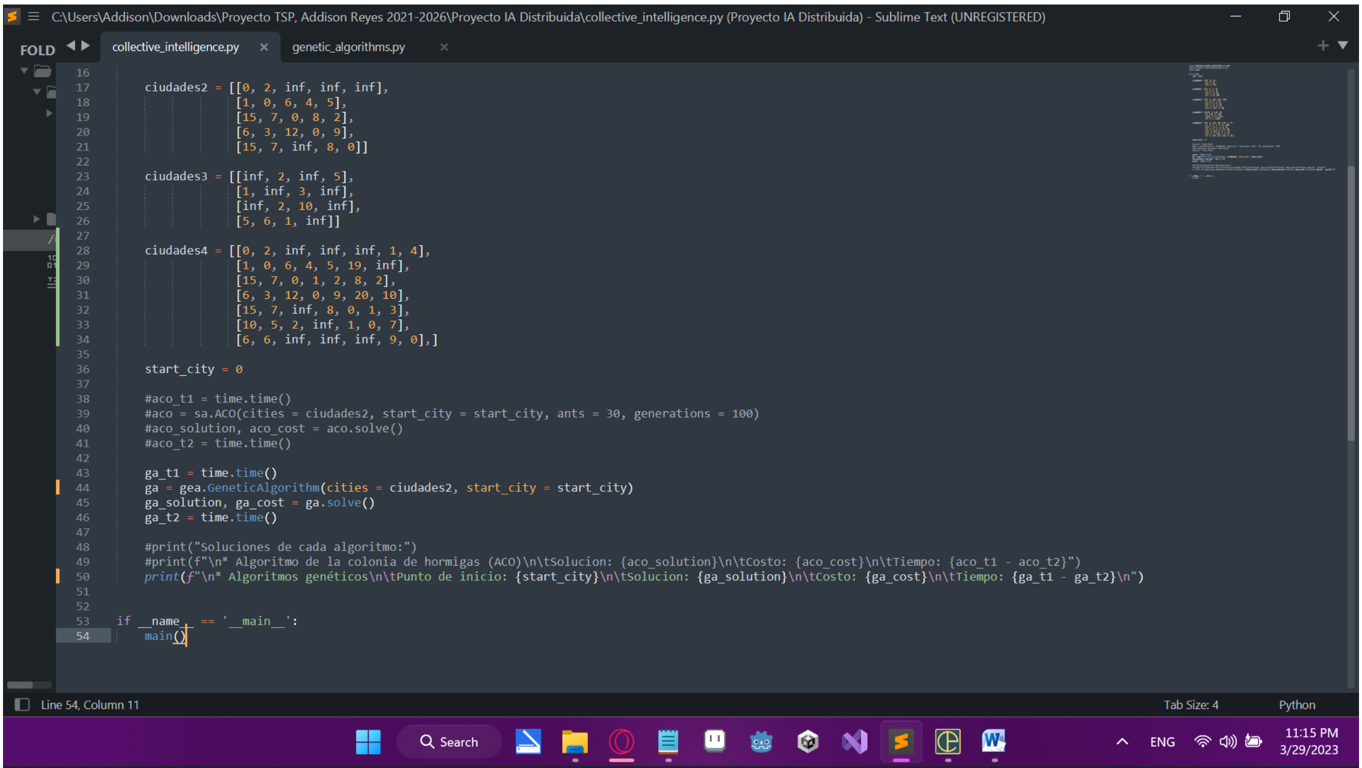
Introducción

El proyecto trata acerca del problema del vendedor ambulante, por sus siglas en ingles TSP, el problema consiste en un vendedor ambulante que tiene que pasar por ***n*** ciudades sin repetir ciudad y terminar la ruta en su punto de origen. El proyecto se enfoca en resolver este problema utilizando algoritmos genéticos python.

Código




```python
import hatchery.genetic_algorithms as gea
#import hatchery.swarm_algorithms as sa
import time

def main():
    inf = 10e9

    ciudades0 = [[0, 10, 5],
                 [2, 0, 7],
                 [10, 3, 0]]

    ciudades1 = [[0, 8, 9, 1],
                 [1, 0, 6, 4],
                 [15, 1, 0, 8],
                 [6, 3, 16, 0]]

    ciudades2 = [[0, 2, inf, inf, inf],
                 [1, 0, 6, 4, 5],
                 [15, 7, 0, 8, 2],
                 [6, 3, 12, 0, 9],
                 [15, 7, inf, 8, 0]]

    ciudades3 = [[inf, 2, inf, 5],
                 [1, inf, 3, inf],
                 [inf, 2, 10, inf],
                 [5, 6, 1, inf]]

    ciudades4 = [[0, 2, inf, inf, inf, 1, 4],
                 [1, 0, 6, 4, 5, 19, inf],
                 [15, 7, 0, 1, 2, 8, 2],
                 [6, 3, 12, 0, 9, 20, 10],
                 [15, 7, inf, 8, 0, 1, 3],
                 [10, 5, 2, inf, 1, 0, 7],
                 [6, 6, inf, inf, inf, 9, 0],]

    start_city = 0

    #aco_t1 = time.time()
    #aco = sa.ACO(cities = ciudades2, start_city = start_city, ants = 30, generations = 100)
    #aco_solution, aco_cost = aco.solve()
    #aco_t2 = time.time()
```



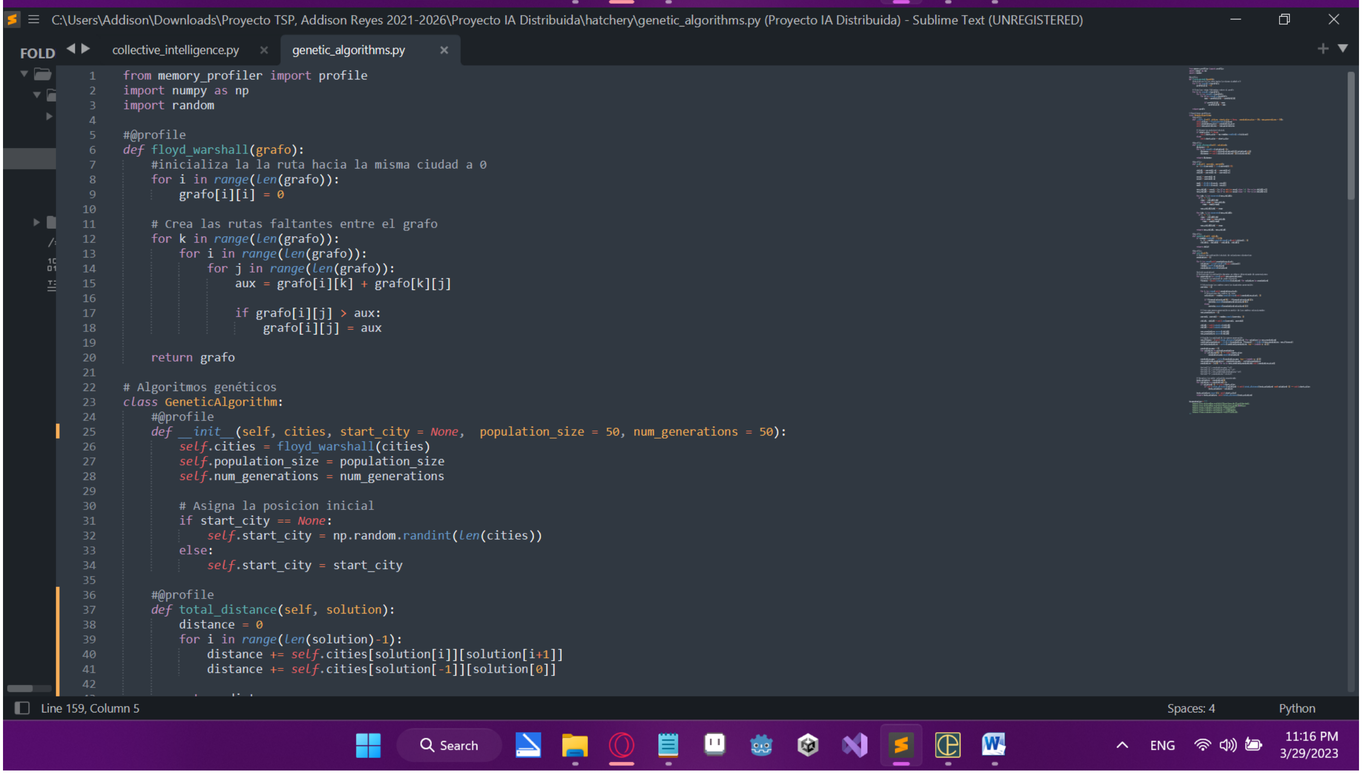
C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\collective_intelligence.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

collective_intelligence.py | genetic_algorithms.py

```python

    ciudades2 = [[0, 2, inf, inf, inf],
                 [1, 0, 6, 4, 5],
                 [15, 7, 0, 8, 2],
                 [6, 3, 12, 0, 9],
                 [15, 7, inf, 8, 0]]

    ciudades3 = [[inf, 2, inf, 5],
                 [1, inf, 3, inf],
                 [inf, 2, 10, inf],
                 [5, 6, 1, inf]]

    ciudades4 = [[0, 2, inf, inf, inf, 1, 4],
                 [1, 0, 6, 4, 5, 19, inf],
                 [15, 7, 0, 1, 2, 8, 2],
                 [6, 3, 12, 0, 9, 20, 10],
                 [15, 7, inf, 8, 0, 1, 3],
                 [10, 5, 2, inf, 1, 0, 7],
                 [6, 6, inf, inf, inf, 9, 0],]

    start_city = 0

    #aco_t1 = time.time()
    #aco = sa.ACO(cities = ciudades2, start_city = start_city, ants = 30, generations = 100)
    #aco_solution, aco_cost = aco.solve()
    #aco_t2 = time.time()

    ga_t1 = time.time()
    ga = gea.GeneticAlgorithm(cities = ciudades2, start_city = start_city)
    ga_solution, ga_cost = ga.solve()
    ga_t2 = time.time()

    #print("Soluciones de cada algoritmo:")
    #print(f"\n* Algoritmo de la colonia de hormigas (ACO)\n\tSolucion: {aco_solution}\n\tCosto: {aco_cost}\n\tTiempo: {aco_t1 - aco_t2}")
    print(f"\n* Algoritmos genéticos\n\tPunto de inicio: {start_city}\n\tSolucion: {ga_solution}\n\tCosto: {ga_cost}\n\tTiempo: {ga_t1 - ga_t2}\n")


if __name__ == '__main__':
    main()
```

Line 54, Column 11 | Tab Size: 4 | Python

Search | ENG | 11:15 PM 3/29/2023



C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

collective_intelligence.py | genetic_algorithms.py

```python
from memory_profiler import profile
import numpy as np
import random

#@profile
def floyd_warshall(grafo):
    #inicializa la la ruta hacia la misma ciudad a 0
    for i in range(len(grafo)):
        grafo[i][i] = 0

    # Crea las rutas faltantes entre el grafo
    for k in range(len(grafo)):
        for i in range(len(grafo)):
            for j in range(len(grafo)):
                aux = grafo[i][k] + grafo[k][j]

                if grafo[i][j] > aux:
                    grafo[i][j] = aux

    return grafo

# Algoritmos genéticos
class GeneticAlgorithm:
    #@profile
    def __init__(self, cities, start_city = None,  population_size = 50, num_generations = 50):
        self.cities = floyd_warshall(cities)
        self.population_size = population_size
        self.num_generations = num_generations

        # Asigna la posicion inicial
        if start_city == None:
            self.start_city = np.random.randint(len(cities))
        else:
            self.start_city = start_city

    #@profile
    def total_distance(self, solution):
        distance = 0
        for i in range(len(solution)-1):
            distance += self.cities[solution[i]][solution[i+1]]
            distance += self.cities[solution[-1]][solution[0]]

```

Line 159, Column 5 | Spaces: 4 | Python

Search | ENG | 11:16 PM 3/29/2023

C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

FOLD collective_intelligence.py genetic_algorithms.py

```python
            self.start_city = start_city

    #@profile
    def total_distance(self, solution):
        distance = 0
        for i in range(len(solution)-1):
            distance += self.cities[solution[i]][solution[i+1]]
            distance += self.cities[solution[-1]][solution[0]]

        return distance

    #@profile
    def pmx(self, parent1, parent2):
        n = (len(parent1) | len(parent2))//2

        child1 = parent1[:n] + parent2[n:]
        child2 = parent2[:n] + parent1[n:]

        secc1 = parent1[:n]
        secc2 = parent2[:n]

        map1 = dict(zip(secc1, secc2))
        map2 = dict(zip(secc2, secc1))

        new_child1 = secc1 + [x if x not in secc1 else 'x' for x in child1[n:]]
        new_child2 = secc2 + [x if x not in secc2 else 'x' for x in child2[n:]]

        for idx, i in enumerate(new_child1):
          if i == 'x':
            vmap = child1[idx]
            while vmap in new_child1:
              vmap = map1[vmap]

            new_child1[idx] = vmap

        for idx, i in enumerate(new_child2):
          if i == 'x':
            vmap = child2[idx]
            while vmap in new_child2:
              vmap = map2[vmap]

            new_child2[idx] = vmap
```

Line 159, Column 5 Spaces: 4 Python

Search ENG 11:16 PM 3/29/2023

C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

FOLD collective_intelligence.py genetic_algorithms.py

```python
            new_child1[idx] = vmap

        for idx, i in enumerate(new_child2):
          if i == 'x':
            vmap = child2[idx]
            while vmap in new_child2:
              vmap = map2[vmap]

            new_child2[idx] = vmap

        return new_child1, new_child2

    #@profile
    def mutation(self, child):
        if random.random() < 0.05:
            i, j = random.sample(range(len(self.cities)), 2)
            child[i], child[j] = child[j], child[i]

        return child

    #@profile
    def solve(self):
        # Genera una población inicial de soluciones aleatorias
        population = []

        for i in range(self.population_size):
            solution = list(range(len(self.cities)))
            random.shuffle(solution)
            population.append(solution)

        #print(population)
        # Evoluciona la población durante un número determinado de generaciones
        for generation in range(self.num_generations):
            # Evalúa la aptitud de cada solución
            fitness = [self.total_distance(solution) for solution in population]

            # Selecciona los padres para la siguiente generación
            parents = []

            for i in range(self.population_size):
                # Selecciona dos padres al azar
                selection = random.sample(range(self.population_size), 2)
```

Line 159, Column 5 Spaces: 4 Python

Search ENG 11:16 PM 3/29/2023

C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

FOLD | collective_intelligence.py | genetic_algorithms.py

```python
        return child

    #@profile
    def solve(self):
        # Genera una población inicial de soluciones aleatorias
        population = []

        for i in range(self.population_size):
            solution = list(range(len(self.cities)))
            random.shuffle(solution)
            population.append(solution)

        #print(population)
        # Evoluciona la población durante un número determinado de generaciones
        for generation in range(self.num_generations):
            # Evalúa la aptitud de cada solución
            fitness = [self.total_distance(solution) for solution in population]

            # Selecciona los padres para la siguiente generación
            parents = []

            for i in range(self.population_size):
                # Selecciona dos padres al azar
                selection = random.sample(range(self.population_size), 2)

                if fitness[selection[0]] > fitness[selection[1]]:
                    parents.append(population[selection[0]])
                else:
                    parents.append(population[selection[1]])

            # Crea una nueva generación a partir de los padres seleccionados
            new_population = []

            parent1, parent2 = random.sample(parents, 2)

            child1, child2 = self.pmx(parent1, parent2)

            child1 = self.mutation(child1)
            child2 = self.mutation(child2)

            new_population.append(child1)
            new_population.append(child2)
```

Line 159, Column 5 | Spaces: 4 | Python

Search | ENG | 11:16 PM 3/29/2023

C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py (Proyecto IA Distribuida) - Sublime Text (UNREGISTERED)

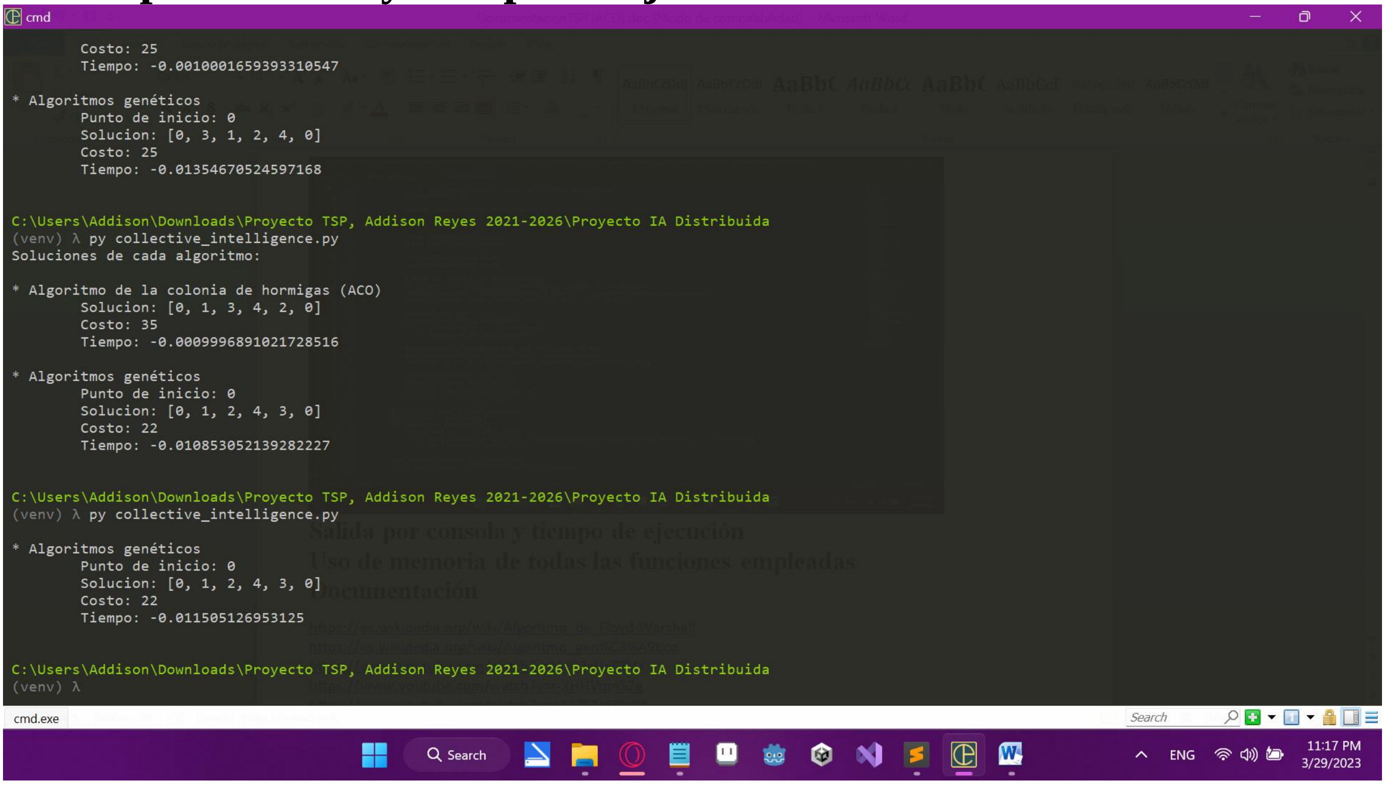
FOLD | collective_intelligence.py | genetic_algorithms.py

```python
            # Crea una nueva generación a partir de los padres seleccionados
            new_population = []

            parent1, parent2 = random.sample(parents, 2)

            child1, child2 = self.pmx(parent1, parent2)

            child1 = self.mutation(child1)
            child2 = self.mutation(child2)

            new_population.append(child1)
            new_population.append(child2)

            # Evalúa la aptitud de la nueva generación
            new_fitness = [self.total_distance(solution) for solution in new_population]
            combined_population = list(zip(population, fitness)) + list(zip(new_population, new_fitness))
            sorted_population = sorted(combined_population, key = lambda x: x[1])

            population_aux = []
            for solution in sorted_population:
                if solution[0][-1] == self.start_city:
                    population_aux.append(solution)

            population_aux = sorted(population_aux, key = lambda x: x[1])
            new_combined_population = population_aux + sorted_population
            population = [x[0] for x in new_combined_population[:self.population_size]]

            #print("1:",population_aux,"\n")
            #print("2:",sorted_population,"\n")
            #print("3:",new_combined_population,"\n")
            #print("4:",population,"\n\n\n")

        # Devuelve la mejor solución encontrada
        best_solution = population[0]
        for solution in population[1:]:
            if solution[-1] == self.start_city:
                if self.total_distance(solution) < self.total_distance(best_solution) and solution[-1] == self.start_city:
                    best_solution = solution

        best_solution.insert(0, self.start_city)
        return best_solution, self.total_distance(best_solution)
```

Line 159, Column 5 | Spaces: 4 | Python

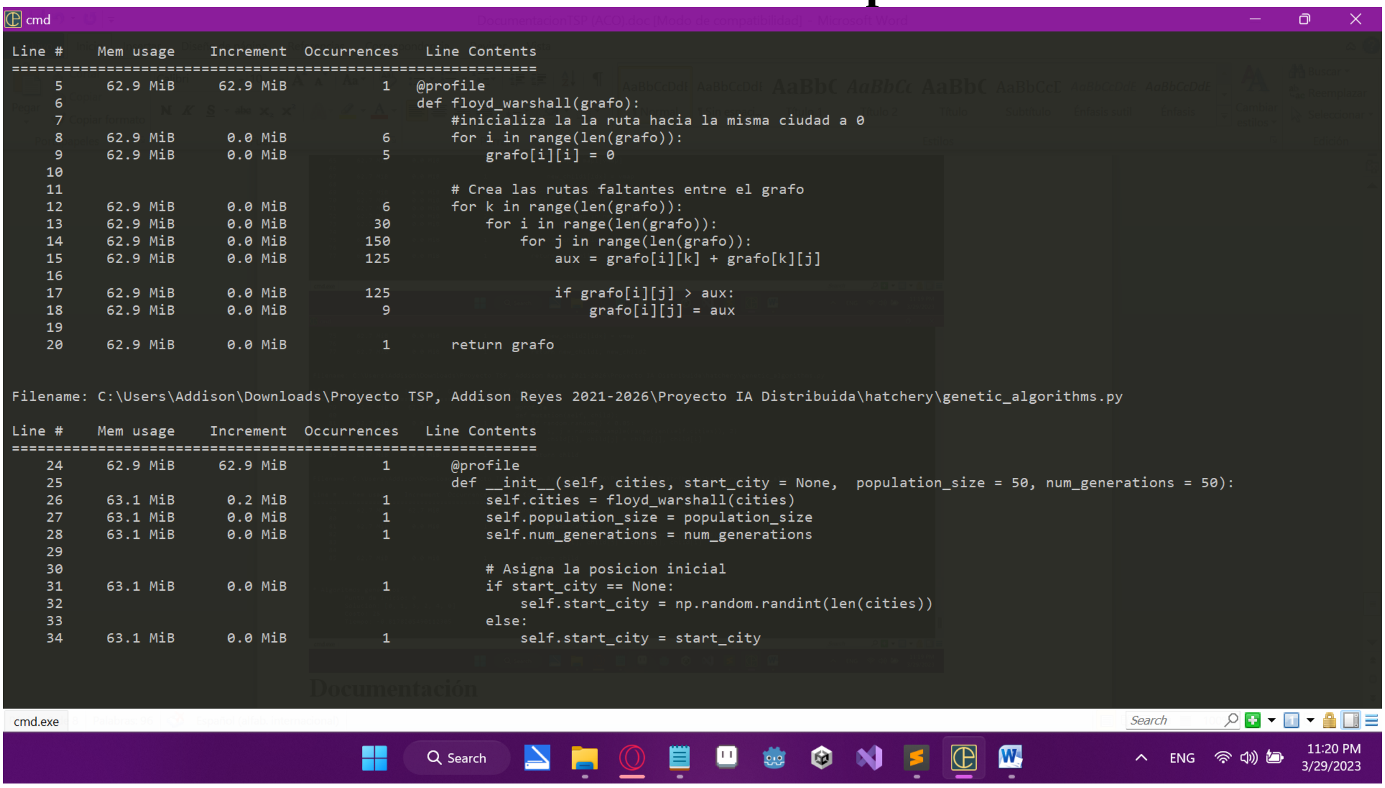
Search | ENG | 11:16 PM 3/29/2023

Salida por consola y tiempo de ejecución



```
        Costo: 25
        Tiempo: -0.0010001659393310547

* Algoritmos genéticos
        Punto de inicio: 0
        Solucion: [0, 3, 1, 2, 4, 0]
        Costo: 25
        Tiempo: -0.01354670524597168


C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida
(venv) λ py collective_intelligence.py
Soluciones de cada algoritmo:

* Algoritmo de la colonia de hormigas (ACO)
        Solucion: [0, 1, 3, 4, 2, 0]
        Costo: 35
        Tiempo: -0.0009996891021728516

* Algoritmos genéticos
        Punto de inicio: 0
        Solucion: [0, 1, 2, 4, 3, 0]
        Costo: 22
        Tiempo: -0.010853052139282227


C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida
(venv) λ py collective_intelligence.py

* Algoritmos genéticos
        Punto de inicio: 0
        Solucion: [0, 1, 2, 4, 3, 0]
        Costo: 22
        Tiempo: -0.011505126953125


C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida
(venv) λ
```

Uso de memoria de todas las funciones empleadas



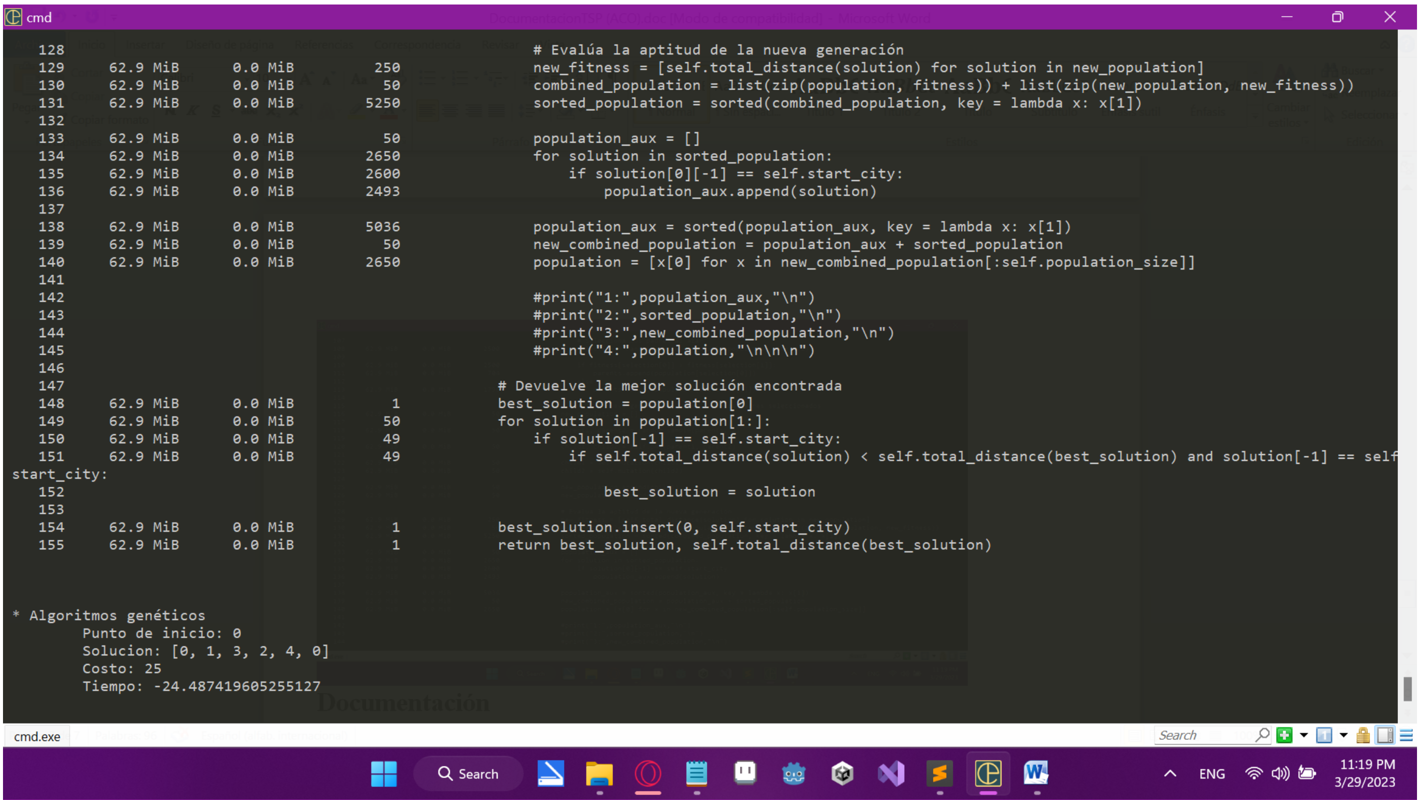
```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     62.9 MiB     62.9 MiB           1   @profile
     6                                         def floyd_warshall(grafo):
     7                                             #inicializa la la ruta hacia la misma ciudad a 0
     8     62.9 MiB      0.0 MiB           6       for i in range(len(grafo)):
     9     62.9 MiB      0.0 MiB           5           grafo[i][i] = 0
    10
    11                                             # Crea las rutas faltantes entre el grafo
    12     62.9 MiB      0.0 MiB           6       for k in range(len(grafo)):
    13     62.9 MiB      0.0 MiB          30           for i in range(len(grafo)):
    14     62.9 MiB      0.0 MiB         150               for j in range(len(grafo)):
    15     62.9 MiB      0.0 MiB         125                   aux = grafo[i][k] + grafo[k][j]
    16
    17     62.9 MiB      0.0 MiB         125                   if grafo[i][j] > aux:
    18     62.9 MiB      0.0 MiB           9                       grafo[i][j] = aux
    19
    20     62.9 MiB      0.0 MiB           1       return grafo


Filename: C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    24     62.9 MiB     62.9 MiB           1       @profile
    25                                             def __init__(self, cities, start_city = None,  population_size = 50, num_generations = 50):
    26     63.1 MiB      0.2 MiB           1           self.cities = floyd_warshall(cities)
    27     63.1 MiB      0.0 MiB           1           self.population_size = population_size
    28     63.1 MiB      0.0 MiB           1           self.num_generations = num_generations
    29
    30                                                 # Asigna la posicion inicial
    31     63.1 MiB      0.0 MiB           1           if start_city == None:
    32                                                     self.start_city = np.random.randint(len(cities))
    33                                                 else:
    34     63.1 MiB      0.0 MiB           1               self.start_city = start_city
```

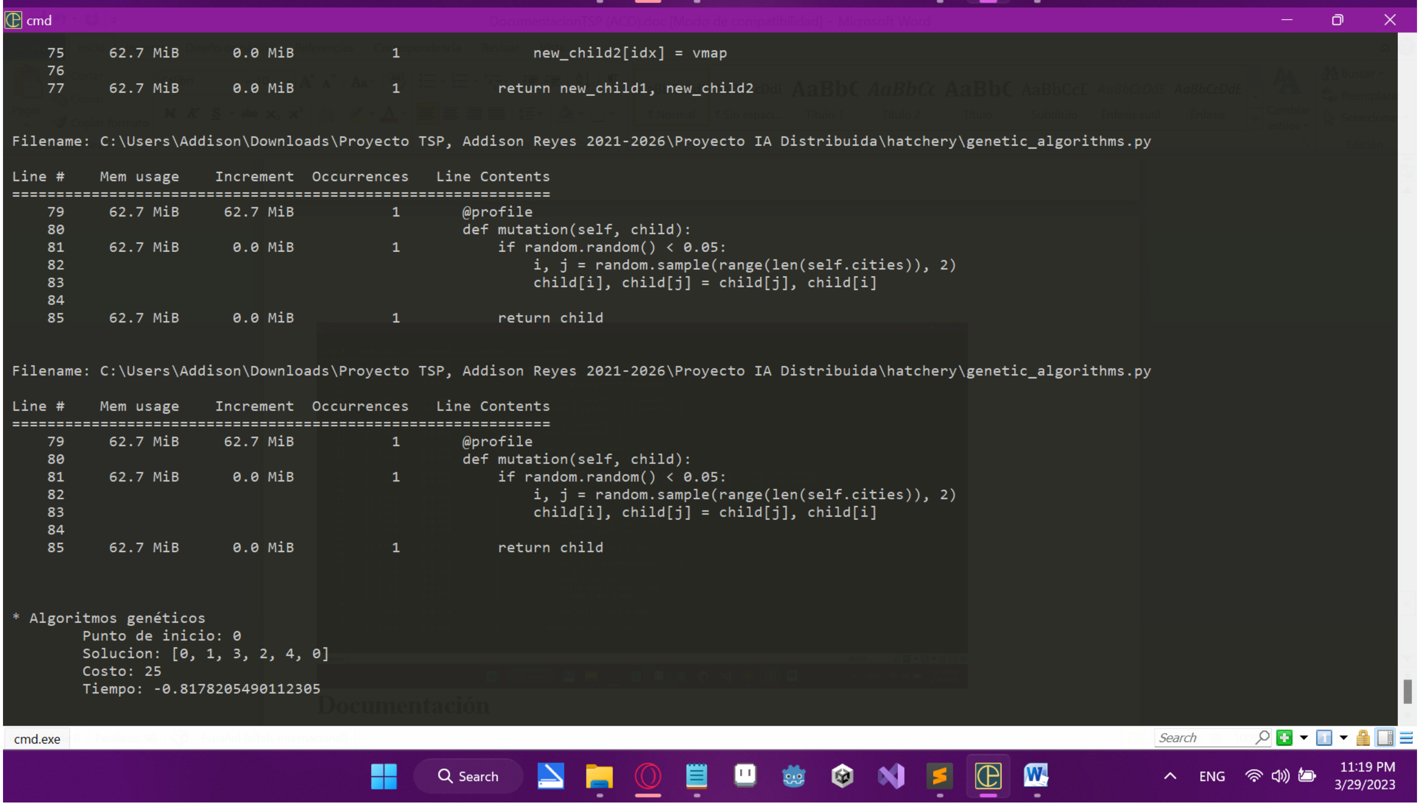
```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    45     62.7 MiB     62.7 MiB           1       @profile
    46                                             def pmx(self, parent1, parent2):
    47     62.7 MiB      0.0 MiB           1           n = (len(parent1) | len(parent2))//2
    48
    49     62.7 MiB      0.0 MiB           1           child1 = parent1[:n] + parent2[n:]
    50     62.7 MiB      0.0 MiB           1           child2 = parent2[:n] + parent1[n:]
    51
    52     62.7 MiB      0.0 MiB           1           secc1 = parent1[:n]
    53     62.7 MiB      0.0 MiB           1           secc2 = parent2[:n]
    54
    55     62.7 MiB      0.0 MiB           1           map1 = dict(zip(secc1, secc2))
    56     62.7 MiB      0.0 MiB           1           map2 = dict(zip(secc2, secc1))
    57
    58     62.7 MiB      0.0 MiB           6           new_child1 = secc1 + [x if x not in secc1 else 'x' for x in child1[n:]]
    59     62.7 MiB      0.0 MiB           6           new_child2 = secc2 + [x if x not in secc2 else 'x' for x in child2[n:]]
    60
    61     62.7 MiB      0.0 MiB           6           for idx, i in enumerate(new_child1):
    62     62.7 MiB      0.0 MiB           5             if i == 'x':
    63     62.7 MiB      0.0 MiB           1               vmap = child1[idx]
    64     62.7 MiB      0.0 MiB           3               while vmap in new_child1:
    65     62.7 MiB      0.0 MiB           2                 vmap = map1[vmap]
    66
    67     62.7 MiB      0.0 MiB           1               new_child1[idx] = vmap
    68
    69     62.7 MiB      0.0 MiB           6           for idx, i in enumerate(new_child2):
    70     62.7 MiB      0.0 MiB           5             if i == 'x':
    71     62.7 MiB      0.0 MiB           1               vmap = child2[idx]
    72     62.7 MiB      0.0 MiB           3               while vmap in new_child2:
    73     62.7 MiB      0.0 MiB           2                 vmap = map2[vmap]
    74
    75     62.7 MiB      0.0 MiB           1               new_child2[idx] = vmap
    76
    77     62.7 MiB      0.0 MiB           1           return new_child1, new_child2
```

```
    42
    43     62.9 MiB      0.0 MiB           1           return distance


Filename: C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    36     62.9 MiB     62.9 MiB           1       @profile
    37                                             def total_distance(self, solution):
    38     62.9 MiB      0.0 MiB           1           distance = 0
    39     62.9 MiB      0.0 MiB           6           for i in range(len(solution)-1):
    40     62.9 MiB      0.0 MiB           5               distance += self.cities[solution[i]][solution[i+1]]
    41     62.9 MiB      0.0 MiB           5               distance += self.cities[solution[-1]][solution[0]]
    42
    43     62.9 MiB      0.0 MiB           1           return distance


Filename: C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    87     62.9 MiB     62.9 MiB           1       @profile
    88                                             def solve(self):
    89                                                 # Genera una población inicial de soluciones aleatorias
    90     62.9 MiB      0.0 MiB           1           population = []
    91
    92     62.9 MiB      0.0 MiB          51           for i in range(self.population_size):
    93     62.9 MiB      0.0 MiB          50               solution = list(range(len(self.cities)))
    94     62.9 MiB      0.0 MiB          50               random.shuffle(solution)
    95     62.9 MiB      0.0 MiB          50               population.append(solution)
    96
    97                                                 #print(population)
    98                                                 # Evoluciona la población durante un número determinado de generaciones
    99     62.9 MiB      0.0 MiB          51           for generation in range(self.num_generations):
   100                                                     # Evalúa la aptitud de cada solución
   101     62.9 MiB      0.0 MiB        2650               fitness = [self.total_distance(solution) for solution in population]
   102
```

```
Filename: C:\Users\Addison\Downloads\Proyecto TSP, Addison Reyes 2021-2026\Proyecto IA Distribuida\hatchery\genetic_algorithms.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    87     62.9 MiB     62.9 MiB           1       @profile
    88                                             def solve(self):
    89                                                 # Genera una población inicial de soluciones aleatorias
    90     62.9 MiB      0.0 MiB           1           population = []
    91
    92     62.9 MiB      0.0 MiB          51           for i in range(self.population_size):
    93     62.9 MiB      0.0 MiB          50               solution = list(range(len(self.cities)))
    94     62.9 MiB      0.0 MiB          50               random.shuffle(solution)
    95     62.9 MiB      0.0 MiB          50               population.append(solution)
    96
    97                                                 #print(population)
    98                                                 # Evoluciona la población durante un número determinado de generaciones
    99     62.9 MiB      0.0 MiB          51           for generation in range(self.num_generations):
   100                                                     # Evalúa la aptitud de cada solución
   101     62.9 MiB      0.0 MiB        2650               fitness = [self.total_distance(solution) for solution in population]
   102
   103                                                     # Selecciona los padres para la siguiente generación
   104     62.9 MiB      0.0 MiB          50               parents = []
   105
   106     62.9 MiB      0.0 MiB        2550               for i in range(self.population_size):
   107                                                         # Selecciona dos padres al azar
   108     62.9 MiB      0.0 MiB        2500                   selection = random.sample(range(self.population_size), 2)
   109
   110     62.9 MiB      0.0 MiB        2500                   if fitness[selection[0]] > fitness[selection[1]]:
   111     62.9 MiB      0.0 MiB         704                       parents.append(population[selection[0]])
   112                                                         else:
   113     62.9 MiB      0.0 MiB        1796                       parents.append(population[selection[1]])
   114
   115                                                     # Crea una nueva generación a partir de los padres seleccionados
   116     62.9 MiB      0.0 MiB          50               new_population = []
   117
   118     62.9 MiB      0.0 MiB          50               parent1, parent2 = random.sample(parents, 2)
   119
   120     62.9 MiB      0.0 MiB          50               child1, child2 = self.pmx(parent1, parent2)
```

```
   107                                                         # Selecciona dos padres al azar
   108     62.9 MiB      0.0 MiB        2500                   selection = random.sample(range(self.population_size), 2)
   109
   110     62.9 MiB      0.0 MiB        2500                   if fitness[selection[0]] > fitness[selection[1]]:
   111     62.9 MiB      0.0 MiB         704                       parents.append(population[selection[0]])
   112                                                         else:
   113     62.9 MiB      0.0 MiB        1796                       parents.append(population[selection[1]])
   114
   115                                                     # Crea una nueva generación a partir de los padres seleccionados
   116     62.9 MiB      0.0 MiB          50               new_population = []
   117
   118     62.9 MiB      0.0 MiB          50               parent1, parent2 = random.sample(parents, 2)
   119
   120     62.9 MiB      0.0 MiB          50               child1, child2 = self.pmx(parent1, parent2)
   121
   122     62.9 MiB      0.0 MiB          50               child1 = self.mutation(child1)
   123     62.9 MiB      0.0 MiB          50               child2 = self.mutation(child2)
   124
   125     62.9 MiB      0.0 MiB          50               new_population.append(child1)
   126     62.9 MiB      0.0 MiB          50               new_population.append(child2)
   127
   128                                                     # Evalúa la aptitud de la nueva generación
   129     62.9 MiB      0.0 MiB         250               new_fitness = [self.total_distance(solution) for solution in new_population]
   130     62.9 MiB      0.0 MiB          50               combined_population = list(zip(population, fitness)) + list(zip(new_population, new_fitness))
   131     62.9 MiB      0.0 MiB        5250               sorted_population = sorted(combined_population, key = lambda x: x[1])
   132
   133     62.9 MiB      0.0 MiB          50               population_aux = []
   134     62.9 MiB      0.0 MiB        2650               for solution in sorted_population:
   135     62.9 MiB      0.0 MiB        2600                   if solution[0][-1] == self.start_city:
   136     62.9 MiB      0.0 MiB        2493                       population_aux.append(solution)
   137
   138     62.9 MiB      0.0 MiB        5036               population_aux = sorted(population_aux, key = lambda x: x[1])
   139     62.9 MiB      0.0 MiB          50               new_combined_population = population_aux + sorted_population
   140     62.9 MiB      0.0 MiB        2650               population = [x[0] for x in new_combined_population[:self.population_size]]
   141
   142                                                     #print("1:",population_aux,"\n")
   143                                                     #print("2:",sorted_population,"\n")
   144                                                     #print("3:",new_combined_population,"\n")
```





Documentación

https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall
https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico
https://www.youtube.com/watch?v=HeG2cUp0TVY
https://www.youtube.com/watch?v=r-XHHVqmS2g
https://www.youtube.com/watch?v=p127FZc6Blc&t