ЛР6 Чураков А А Р3231 В-19.
Генетический алгоритм.
Задача коммивояжёра.

---

1. Исходные данные

- Число городов: 5.
- Размер популяции: $N_{\text{pop}} = 4$.
- Вероятность мутации (обмен двух генов): $p_{\text{mut}} = 0{,}01$.
- Матрица расстояний D:

	0	1	2	3	4
0	0	4	6	2	9
1	4	0	3	2	9
2	6	3	0	5	9
3	2	2	5	0	8
4	9	9	9	8	0

2. Поколение 1

2.1 Начальная популяция

№	Маршрут	Длина
1	4 2 1 0 3	26
2	4 2 0 3 1	28
3	4 2 3 1 0	29
4	4 2 0 1 3	29

2.2 Скрещивания и мутации

Пара 1 (индексы $[0, 3]$, точки $1, 2$).

Родитель 1	4 | 2 1 | 0 3
Родитель 2	4 | 2 0 | 1 3
Потомок 1	1 2 0 3 4
Потомок 2	0 2 1 3 4
Мутация	не была выполнена ($p_{\text{mut}} = 0{,}01$).

Пара 2 (индексы $[0, 1]$, точки $1, 2$).

Родитель 1	4 | 21 | 03
Родитель 2	4 | 20 | 31
Потомок 1	12034
Потомок 2	02134
Мутация	не была выполнена.

2.3 Расширенная популяция, отбор лучших 4

ID	Маршрут	Длина
P_1	42103	26
P_2	42013	29
P_3	42103	26
P_4	42031	28
C_1	12034	28
C_2	02134	28
C_3	12034	28
C_4	02134	28

№	Маршрут	Длина
1	42103	26
2	42031	28
3	12034	28
4	02134	28

3. Поколение 2

3.1 Текущая популяция

Та же, что после отбора в конце поколения 1.

№	Маршрут	Длина
1	42103	26
2	42031	28
3	12034	28
4	02134	28

3.2 Скрещивания и мутации

Пара 1 (индексы $[1, 0]$, точки $1, 3$).

Родитель 1	4 | 203 | 1
Родитель 2	4 | 210 | 3
Потомок 1	32104
Потомок 2	12034
Мутация	не выполнена.

Пара 2 (индексы $[2, 3]$, точки $1, 2$).

Родитель 1	1 | 2 0 | 3 4
Родитель 2	0 | 2 1 | 3 4
Потомок 1	0 2 1 3 4
Потомок 2	1 2 0 3 4
Мутация	не выполнена.

ID	Маршрут	Длина
P_1	4 2 0 3 1	28
P_2	4 2 1 0 3	26
P_3	1 2 0 3 4	28
P_4	0 2 1 3 4	28
C_5	3 2 1 0 4	29
C_6	1 2 0 3 4	28
C_7	0 2 1 3 4	28
C_8	1 2 0 3 4	28

3.3 Популяция после отбора

№	Маршрут	Длина
1	4 2 1 0 3	26
2	4 2 0 3 1	28
3	1 2 0 3 4	28
4	0 2 1 3 4	28

4. Поколение 3

4.1 Текущая популяция

Идентична популяции конца поколения 2.

4.2 Скрещивания и мутации

Пара 1 (индексы $[0, 2]$, точки $2, 3$).

Родитель 1	4 2 | 1 0 | 3
Родитель 2	1 2 | 0 3 | 4
Потомок 1	4 2 0 3 1
Потомок 2	3 4 1 0 2
Мутация	не выполнена.

Пара 2 (индексы $[3, 2]$, точки $1, 3$).

Родитель 1	0 | 2 1 3 | 4
Родитель 2	1 | 2 0 3 | 4
Потомок 1	1 2 0 3 4

Потомок 2	0 2 1 3 4
Мутация	не выполнена.

ID	Маршрут	Длина
P_1	4 2 1 0 3	26
P_2	1 2 0 3 4	28
P_3	0 2 1 3 4	28
P_4	1 2 0 3 4	28
C_9	4 2 0 3 1	28
C_10	3 4 1 0 2	32
C_11	1 2 0 3 4	28
C_12	0 2 1 3 4	28

4.3 Итоговая популяция (после отбора)

№	Маршрут	Длина
1	4 2 1 0 3	26
2	4 2 0 3 1	28
3	1 2 0 3 4	28
4	0 2 1 3 4	28

5. Лучший найденный маршрут

- Перестановка (индексация с нуля):
  `[4, 2, 1, 0, 3]`
- В терминах городов 1…5 (добавлено +1):
  5 → 3 → 2 → 1 → 4 → 5.
- Лучшая длина тура за 3 итерации: **26**.

6. Программная реализация

```
import random
import numpy as np

# --------------------------- CONSTANTS ------------------------------------
MUTATION_RATE = 0.01
NUM_CITIES = 5
POP_SIZE = 4
NUM_GENERATIONS = 3

# Distance matrix between the 5 cities
DIST = np.array(
    [
        [0, 4, 6, 2, 9],
        [4, 0, 3, 2, 9],
```

```
        [6, 3, 0, 5, 9],
        [2, 2, 5, 0, 8],
        [9, 9, 9, 8, 0],
    ]
)

# ------------------------------------------------------------------------ #
def path_len(route: list[int]) -> int:
    """
    Compute total tour length (objective value).

    This sums the distances for each consecutive edge in the permutation and
    finally adds the edge that closes the tour (last city  first city).
    """
    return sum(DIST[route[i], route[(i + 1) % NUM_CITIES]]
               for i in range(NUM_CITIES))

# ------------------------------------------------------------------------ #
def init_population() -> list[list[int]]:
    """Create initial population of random permutations."""
    return [random.sample(range(NUM_CITIES), NUM_CITIES)
            for _ in range(POP_SIZE)]

# ------------------------------------------------------------------------ #
def mutate(route: list[int]) -> bool:
    """Swap-mutation: exchange two random positions (returns True if mutated).
        """
    if random.random() < MUTATION_RATE:
        i, j = random.sample(range(NUM_CITIES), 2)
        route[i], route[j] = route[j], route[i]
        return True
    return False

# -------- Order-Crossover with fragment exchange and left-fill ------------ #
def ox(parent_host: list[int], parent_donor: list[int], a: int, b: int) -> list[
    int]:
    """
    1. Exchange step: copy slice parent_donor[a:b] into child
       (this is where offspring inherit the foreign fragment).
    2. Filling step: walk through parent_host (starting at a+1, cyclic)
       and insert missing cities into the LEFT-most free stars,
       then continue on the right side of the exchanged slice.

     The fragment exchange happens once at line 'child[a:b+1] = ...'.
     The subsequent loop fills remaining gaps preserving order
      and ensuring no duplicates.
    """
    n = len(parent_host)
    child = [-1] * n # empty child
    child[a:b+1] = parent_donor[a:b+1] # *** EXCHANGE SEGMENT ***
```

```
    # insertion order: all indices left of slice, then right of slice
    positions = list(range(0, a)) + list(range(b + 1, n))

    idx_host = (a + 1) % n # start scanning host
    for _ in range(n):
        gene = parent_host[idx_host]
        if gene not in child:
            pos = positions.pop(0) # *** FILL FIRST LEFT STAR ***
            child[pos] = gene
        idx_host = (idx_host + 1) % n
    return child

# ------------------------------ GA MAIN LOOP ------------------------------- #
def genetic_algorithm():
    population = init_population()

    for gen in range(NUM_GENERATIONS):
        fitness = np.array([path_len(r) for r in population])
        probs = 1 / fitness
        probs /= probs.sum() # better tour gets more chance to be the parent

        print(f"\nGeneration {gen + 1}:")
        print("Population:", population)
        print("Distances:", fitness)

        new_population = []
        pair_no = 0
        while len(new_population) < POP_SIZE:
            i1, i2 = np.random.choice(len(population), 2, replace=False, p=probs)
            p1, p2 = population[i1], population[i2]
            pair_no += 1

            # inner crossover points (no edge positions)
            a, b = sorted(random.sample(range(1, NUM_CITIES - 1), 2))

            print(f"\nPair {pair_no}: [{i1}, {i2}] cuts {a},{b}")
            print("Parent 1:", *p1[:a], "|", *p1[a:b + 1], "|", *p1[b + 1:])
            print("Parent 2:", *p2[:a], "|", *p2[a:b + 1], "|", *p2[b + 1:])

            child1 = ox(p1, p2, a, b)
            child2 = ox(p2, p1, a, b)
            print("Child 1:", child1)
            print("Child 2:", child2)

            if mutate(child1):
                print("Child 1 MUTATED:", child1)
            if mutate(child2):
                print("Child 2 MUTATED:", child2)

            new_population.extend([child1, child2])
```

```
        # Elitism: join old + new individuals and keep the best 4 tours
        # (lowest objective value) to form the next generation.
        population = sorted(population + new_population, key=path_len)[:POP_SIZE]

        print("\nEnlarged population:", population)
        print("Distances:", [path_len(r) for r in population])

    best = min(population, key=path_len)
    return best, path_len(best)

# ---------------------------------------------------------------------------- #
if __name__ == "__main__":
    best_route, best_dist = genetic_algorithm()
    print(f"\nBest route: {best_route}, Distance: {best_dist}")
```