Questao 9

Codigo Fonte:

```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
import io
import os
from contextlib import redirect_stdout
from fpdf import FPDF
from fpdf.enums import XPos, YPos
import warnings
warnings.filterwarnings('ignore')

# --- Parâmetros do Algoritmo Genético ---
POPULATION_SIZE = 200
GENERATIONS = 3000
MUTATION_RATE = 0.05
CROSSOVER_RATE = 0.9
TOURNAMENT_SIZE = 5
ELITISM_COUNT = 2

def calculate_distance_matrix(cities):
    """Pré-calcula a matriz de distâncias euclidianas entre todas as cidades."""
    num_cities = len(cities)
    dist_matrix = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(i, num_cities):
            dist = math.sqrt((cities[i][0] - cities[j][0])**2 + (cities[i][1] - cities[j][1])**2)
            dist_matrix[i][j] = dist_matrix[j][i] = dist
    return dist_matrix

def calculate_tour_distance(tour, dist_matrix):
    """Calcula a distância total de um tour usando a matriz pré-calculada."""
    total_distance = 0
    for i in range(len(tour)):
        # Distância do ponto atual para o próximo (com wrap around no final)
        from_city = tour[i]
        to_city = tour[(i + 1) % len(tour)]
        total_distance += dist_matrix[from_city][to_city]
    return total_distance

def ordered_crossover(parent1, parent2):
    """Executa o Ordered Crossover (OX1)."""
    size = len(parent1)
    child = [-1] * size
    start, end = sorted(random.sample(range(size), 2))

    # Copia o segmento do pai 1 para o filho
```

```
    child[start:end+1] = parent1[start:end+1]

    # Preenche o resto com os genes do pai 2
    parent2_genes = [item for item in parent2 if item not in child]
    child_idx = (end + 1) % size

    for gene in parent2_genes:
        if child_idx == start: # Pula o segmento já preenchido
            child_idx = (end + 1) % size
        child[child_idx] = gene
        child_idx = (child_idx + 1) % size

    return child

def swap_mutation(tour):
    """Troca a posição de duas cidades aleatórias no tour."""
    idx1, idx2 = random.sample(range(len(tour)), 2)
    tour[idx1], tour[idx2] = tour[idx2], tour[idx1]
    return tour

def plot_tour(cities, best_tour, best_distance):
    """Plota o mapa de cidades e a melhor rota encontrada."""
    ordered_cities = np.array([cities[i] for i in best_tour])

    plt.figure(figsize=(10, 8))
    # Adiciona a primeira cidade ao final para fechar o ciclo no gráfico
    tour_path = np.vstack([ordered_cities, ordered_cities[0]])

    plt.plot(tour_path[:, 0], tour_path[:, 1], 'o-')

    # Anota os números das cidades
    for i, city in enumerate(cities):
        plt.text(city[0] + 0.5, city[1] + 0.5, str(i + 1), color="red", fontsize=12)

    plt.title(f"Melhor Rota Encontrada (Distancia: {best_distance:.2f})")
    plt.xlabel("Coordenada X")
    plt.ylabel("Coordenada Y")
    plt.grid(True)
    plt.savefig("resultado_tour_q9.png")
    plt.close()


def solve_and_capture_output():
    """Função principal que executa o AG para o PCV."""
    print("Questao 9: Algoritmo Genetico para o Problema do Caixeiro Viajante")
    print("=" * 70)

    # 1. Dados do Problema (CORRIGIDOS DE ACORDO COM A IMAGEM NÍTIDA)
    cities = [
        (0, 0),    # Cidade 1
        (3, 27),   # Cidade 2
        (14, 22),  # Cidade 3
        (1, 13),   # Cidade 4
```

```
    (20, 3),   # Cidade 5
    (20, 16),  # Cidade 6
    (28, 12),  # Cidade 7
    (30, 31),  # Cidade 8
    (11, 19),  # Cidade 9
    (7, 3),    # Cidade 10
    (10, 25)   # Cidade 11
]
num_cities = len(cities)
dist_matrix = calculate_distance_matrix(cities)
print(f"Problema com {num_cities} cidades (coordenadas corrigidas).")

# 2. Inicialização da População
population = [random.sample(range(num_cities), num_cities) for _ in range(POPULATION_SIZE)]

best_tour_overall = None
best_distance_overall = float('inf')

print("\nIniciando a busca pela melhor rota...")

# 3. Loop Evolutivo
for generation in range(GENERATIONS):
    # Calcula a aptidão de cada indivíduo
    fitness_scores = [1 / calculate_tour_distance(tour, dist_matrix) for tour in population]

    # Seleciona os melhores para elitismo
    elite_indices = np.argsort(fitness_scores)[-ELITISM_COUNT:]
    new_population = [population[i] for i in elite_indices]

    # Seleção por torneio
    selected_parents = []
    for _ in range(len(population)):
        tournament_indices = np.random.choice(range(len(population)), TOURNAMENT_SIZE)
        tournament_fitness = [fitness_scores[i] for i in tournament_indices]
        winner_index = tournament_indices[np.argmax(tournament_fitness)]
        selected_parents.append(population[winner_index])

    # Gera nova população com Crossover e Mutação
    while len(new_population) < POPULATION_SIZE:
        p1, p2 = random.sample(selected_parents, 2)
        if random.random() < CROSSOVER_RATE:
            child = ordered_crossover(p1, p2)
        else:
            child = p1[:]

        if random.random() < MUTATION_RATE:
            child = swap_mutation(child)

        new_population.append(child)

    population = new_population

    # Acompanha o melhor indivíduo
```

```
        current_best_idx = np.argmax(fitness_scores)
        current_best_dist = calculate_tour_distance(population[current_best_idx], dist_matrix)
        if current_best_dist < best_distance_overall:
            best_distance_overall = current_best_dist
            best_tour_overall = population[current_best_idx]

        if (generation + 1) % 500 == 0:
            print(f"Geracao {generation+1}: Melhor distancia ate agora = {best_distance_overall:.2f}")

    print("\nBusca finalizada.")
    print("\n" + "=" * 70)
    print("Melhor Rota Encontrada:")

    # Ajusta a rota para começar na cidade 1 (índice 0) para melhor visualização
    start_index = best_tour_overall.index(0)
    best_tour_ordered = best_tour_overall[start_index:] + best_tour_overall[:start_index]
    # Adiciona 1 aos índices para corresponder aos nomes das cidades (1 a 11)
    best_tour_named = [city_idx + 1 for city_idx in best_tour_ordered]

    print(f"  - Sequencia de Cidades: {' -> '.join(map(str, best_tour_named))} -> {best_tour_named[0]}")
    print(f"  - Distancia Total da Rota: {best_distance_overall:.4f}")

    # 4. Gera o gráfico da melhor rota
    plot_tour(cities, best_tour_ordered, best_distance_overall)
    print("\nGrafico da melhor rota foi salvo como 'resultado_tour_q9.png'")
    print("=" * 70)

def generate_pdf_report(code_content, output_content):
    """Gera um PDF com o código, o output e o gráfico."""
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Courier", 'B', 16)
    pdf.cell(0, 10, 'Questao 9', new_x=XPos.LMARGIN, new_y=YPos.NEXT, align='C')
    pdf.ln(10)
    pdf.set_font("Courier", 'B', 12)
    pdf.cell(0, 10, 'Codigo Fonte:', new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Courier", '', 8)
    pdf.multi_cell(0, 5, code_content)
    pdf.add_page()
    pdf.set_font("Courier", 'B', 12)
    pdf.cell(0, 10, 'Output da Execucao:', new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Courier", '', 10)
    output_content_safe = output_content.encode('latin-1', 'replace').decode('latin-1')
    pdf.multi_cell(0, 5, output_content_safe)

    # Adiciona a imagem do gráfico
    if os.path.exists("resultado_tour_q9.png"):
        pdf.add_page()
        pdf.set_font("Courier", 'B', 12)
        pdf.cell(0, 10, 'Grafico da Melhor Rota Encontrada:', new_x=XPos.LMARGIN, new_y=YPos.NEXT)
        pdf.image("resultado_tour_q9.png", x=10, y=30, w=190)

    pdf_file_name = "resultado_questao_9.pdf"
```

```
    pdf.output(pdf_file_name)
    return pdf_file_name

# --- Bloco Principal de Execução ---
if __name__ == "__main__":
    output_buffer = io.StringIO()
    with redirect_stdout(output_buffer):
        solve_and_capture_output()
    output_content = output_buffer.getvalue()

    print("--- [INICIO] Resultado da Execucao no Terminal ---")
    print(output_content)
    print("--- [FIM] Resultado da Execucao no Terminal ---")

    try:
        with open(__file__, 'r', encoding='utf-8') as f:
            code_content = f.read()
    except Exception as e:
        code_content = f"Nao foi possivel ler o arquivo do codigo: {e}"

    try:
        pdf_file = generate_pdf_report(code_content, output_content)
        print(f"\nPDF '{pdf_file}' gerado com sucesso no diretorio atual!")
    except Exception as e:
        print(f"\nOcorreu um erro ao gerar o PDF: {e}")
```

Output da Execucao:

```
Questao 9: Algoritmo Genetico para o Problema do Caixeiro Viajante
======================================================================
Problema com 11 cidades (coordenadas corrigidas).

Iniciando a busca pela melhor rota...
Geracao 500: Melhor distancia ate agora = 122.77
Geracao 1000: Melhor distancia ate agora = 122.77
Geracao 1500: Melhor distancia ate agora = 122.77
Geracao 2000: Melhor distancia ate agora = 122.77
Geracao 2500: Melhor distancia ate agora = 122.77
Geracao 3000: Melhor distancia ate agora = 122.77

Busca finalizada.

======================================================================
Melhor Rota Encontrada:
  - Sequencia de Cidades: 1 -> 4 -> 2 -> 11 -> 9 -> 3 -> 8 -> 6 -> 7 -> 5 -> 10 -> 1
  - Distancia Total da Rota: 122.7730

Grafico da melhor rota foi salvo como 'resultado_tour_q9.png'
======================================================================
```

Grafico da Melhor Rota Encontrada: