Questao 6

Codigo Fonte:

```
import numpy as np
import random
import io
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 = 150
GENERATIONS = 300
MUTATION_RATE = 0.2
CROSSOVER_RATE = 0.8
TOURNAMENT_SIZE = 5
ELITISM_COUNT = 2

# --- Funções do Algoritmo Genético ---
# Adaptadas para receber o grafo como parâmetro

def create_random_path(graph, start, end):
    """Gera um caminho aleatório válido (sem ciclos) do início ao fim."""
    path = [start]
    current_node = start
    while current_node != end:
        neighbors = [node for node in graph.get(current_node, {}) if node not in path]
        if not neighbors:
            return None # Beco sem saída
        next_node = random.choice(neighbors)
        path.append(next_node)
        current_node = next_node
    return path

def calculate_distance(graph, path):
    """Calcula a distância total de um caminho."""
    total_distance = 0
    for i in range(len(path) - 1):
        try:
            total_distance += graph[path[i]][path[i+1]]
        except KeyError:
            return float('inf') # Caminho inválido
    return total_distance

def calculate_fitness(graph, path):
    """Calcula a aptidão de um caminho (inverso da distância)."""
    distance = calculate_distance(graph, path)
    if distance == float('inf') or distance == 0:
        return 0
    return 1 / distance

def selection(population, fitness_scores):
    """Seleção por Torneio."""
    selected = []
    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.append(population[winner_index])
    return selected

def crossover(parent1, parent2):
    """Realiza o cruzamento encontrando um ponto em comum."""
    offspring1, offspring2 = parent1[:], parent2[:]
    common_nodes = list(set(parent1[1:-1]) & set(parent2[1:-1]))

    if not common_nodes or random.random() > CROSSOVER_RATE:
        return offspring1, offspring2

    crossover_point = random.choice(common_nodes)
    idx1 = parent1.index(crossover_point)
    idx2 = parent2.index(crossover_point)

    offspring1 = parent1[:idx1] + parent2[idx2:]
    offspring2 = parent2[:idx2] + parent1[idx1:]
    return offspring1, offspring2

def mutate(graph, path, start, end):
    """Realiza a mutação substituindo um subcaminho por um novo."""
    if len(path) <= 2 or random.random() > MUTATION_RATE:
        return path

    start_idx = random.randint(1, len(path) - 2)
    end_idx = random.randint(start_idx, len(path) - 2)
    if start_idx >= end_idx: return path

    start_node, end_node = path[start_idx-1], path[end_idx]
    sub_path = create_random_path(graph, start_node, end_node)

    if sub_path:
        mutated_path = path[:start_idx-1] + sub_path + path[end_idx+1:]
        if mutated_path[0] == start and mutated_path[-1] == end:
            return mutated_path
    return path

def run_genetic_algorithm(graph, start_node, end_node):
    """Executa o ciclo completo do AG para um dado grafo."""
    population = []
    while len(population) < POPULATION_SIZE:
        path = create_random_path(graph, start_node, end_node)
        if path: population.append(path)
    best_path_overall = None
    best_distance_overall = float('inf')

    print(f"Buscando melhor caminho de '{start_node}' para '{end_node}'...")

    for generation in range(GENERATIONS):
        fitness_scores = [calculate_fitness(graph, p) for p in population]
        best_idx_gen = np.argmax(fitness_scores)
        best_distance_gen = calculate_distance(graph, population[best_idx_gen])

        if best_distance_gen < best_distance_overall:
            best_distance_overall = best_distance_gen
            best_path_overall = population[best_idx_gen]

        new_population = []
        elite_indices = np.argsort(fitness_scores)[-ELITISM_COUNT:]
        for i in elite_indices: new_population.append(population[i])

        selected_parents = selection(population, fitness_scores)

        while len(new_population) < POPULATION_SIZE:
            p1, p2 = random.sample(selected_parents, 2)
            o1, o2 = crossover(p1, p2)
            new_population.append(mutate(graph, o1, start_node, end_node))
            if len(new_population) < POPULATION_SIZE:
                new_population.append(mutate(graph, o2, start_node, end_node))

        population = new_population

    print(f"\nMelhor caminho encontrado de '{start_node}' para '{end_node}':")
    print(f"  - Rota: {' -> '.join(best_path_overall)}")
    print(f"  - Distancia Total: {best_distance_overall}")

def solve_and_capture_output():
    """Define os grafos e executa o GA para cada parte da questão."""
    print("Questao 6: Algoritmo Genetico para Caminho Mais Curto")
    print("=" * 60)

    # --- Parte (a) ---
    print("\n--- Parte (a) ---\n")

    graph_a = {
        'A': {'B': 2, 'G': 6},
        'B': {'A': 2, 'C': 7, 'E': 2},
        'C': {'B': 7, 'D': 3, 'F': 3},
        'D': {'C': 3, 'H': 2},
        'E': {'B': 2, 'F': 2, 'G': 1},
        'F': {'C': 3, 'E': 2, 'H': 2},
        'G': {'A': 6, 'E': 1, 'H': 4},
        'H': {'D': 2, 'F': 2, 'G': 4}
    }
    run_genetic_algorithm(graph_a, start_node='A', end_node='D')
    print("\n" + "=" * 60)

    # --- Parte (b) ---
    print("\n--- Parte (b) ---\n")

    # Grafo (b) com o mapeamento final fornecido pelo usuário.
    graph_b = {
        'a': {'b': 16, 'c': 10, 'd': 5},
        'b': {'a': 16, 'c': 2, 'f': 4, 'g': 6},
        'c': {'a': 10, 'b': 2, 'd': 4, 'f': 12},
        'd': {'a': 5, 'c': 4, 'e': 15},
        'e': {'d': 15, 'f': 3, 'z': 5},
        'f': {'b': 4, 'c': 12, 'e': 3, 'g': 8, 'z': 16},
        'g': {'b': 6, 'f': 8, 'z': 7},
        'z': {'e': 5, 'f': 16, 'g': 7}
    }
    run_genetic_algorithm(graph_b, start_node='a', end_node='z')
    print("\n" + "=" * 60)

def generate_pdf_report(code_content, output_content):
    """Gera um PDF com o código e o output."""
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Courier", 'B', 16)
    pdf.cell(0, 10, 'Questao 6', 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)
    pdf_file_name = "resultado_questao_6.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 6: Algoritmo Genetico para Caminho Mais Curto
============================================================

--- Parte (a) ---

Buscando melhor caminho de 'A' para 'D'...

Melhor caminho encontrado de 'A' para 'D':
  - Rota: A -> B -> E -> F -> H -> D
  - Distancia Total: 10

============================================================

--- Parte (b) ---

Buscando melhor caminho de 'a' para 'z'...

Melhor caminho encontrado de 'a' para 'z':
  - Rota: a -> d -> c -> b -> f -> e -> z
  - Distancia Total: 23

============================================================
```