Questao 1

Codigo Fonte:

```
import random
import numpy as np
import io
import os
from contextlib import redirect_stdout
from fpdf import FPDF

# --- Definição do Grafo (Mapa) ---
MAPA_CIDADES = {
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Zerind': {'Oradea': 71, 'Arad': 75},
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Fagaras': {'Sibiu': 99, 'Bucareste': 211},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucareste': 101},
    'Bucareste': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},
    'Giurgiu': {'Bucareste': 90},
    'Urziceni': {'Bucareste': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Eforie': {'Hirsova': 86},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}

# --- 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 (sem alterações) ---

def create_random_path(start, end):
    path = [start]
    current_city = start
    while current_city != end:
        neighbors = [city for city in MAPA_CIDADES[current_city] if city not in path]
        if not neighbors: return None
```

```
        next_city = random.choice(neighbors)
        path.append(next_city)
        current_city = next_city
    return path

def calculate_distance(path):
    total_distance = 0
    for i in range(len(path) - 1):
        try:
            total_distance += MAPA_CIDADES[path[i]][path[i+1]]
        except KeyError:
            return float('inf')
    return total_distance

def calculate_fitness(path):
    distance = calculate_distance(path)
    if distance == float('inf') or distance == 0: return 0
    return 1 / distance

def selection(population, fitness_scores):
    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):
    offspring1, offspring2 = parent1[:], parent2[:]
    common_cities = list(set(parent1[1:-1]) & set(parent2[1:-1]))
    if not common_cities or random.random() > CROSSOVER_RATE:
        return offspring1, offspring2
    crossover_point = random.choice(common_cities)
    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(path):
    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(start_node, end_node)
    if sub_path:
        mutated_path = path[:start_idx-1] + sub_path + path[end_idx+1:]
        if mutated_path[0] == path[0] and mutated_path[-1] == path[-1]:
            return mutated_path
    return path
```

```
def run_genetic_algorithm(start_city, end_city):
    population = []
    while len(population) < POPULATION_SIZE:
        path = create_random_path(start_city, end_city)
        if path: population.append(path)

    best_path_overall = None
    best_distance_overall = float('inf')

    print(f"Buscando melhor caminho de {start_city} para {end_city}...")
    for generation in range(GENERATIONS):
        fitness_scores = [calculate_fitness(p) for p in population]
        best_idx_gen = np.argmax(fitness_scores)
        best_distance_gen = calculate_distance(population[best_idx_gen])

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

        if (generation + 1) % 50 == 0:
             print(f"Geracao {generation+1}: Melhor distancia = {best_distance_overall}")

        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:
            parent1, parent2 = random.sample(selected_parents, 2)
            offspring1, offspring2 = crossover(parent1, parent2)
            new_population.append(mutate(offspring1))
            if len(new_population) < POPULATION_SIZE:
                new_population.append(mutate(offspring2))

        population = new_population

    return best_path_overall, best_distance_overall

def execute_and_capture_output():
    """
    Executa a lógica principal e captura tudo que seria impresso no console.
    """
    print("--- Questao 1: Algoritmo Genetico para Caminho Mais Curto ---\n")

    # a) Oradea e Eforie
    path_a, dist_a = run_genetic_algorithm('Oradea', 'Eforie')
    print("\n--- Resultado para (a) Oradea -> Eforie ---")
    print(f"Melhor caminho encontrado: {' -> '.join(path_a)}")
    print(f"Distancia total: {dist_a} km")

    print("\n" + "="*50 + "\n")
```

```
    # b) Arad e Neamt
    path_b, dist_b = run_genetic_algorithm('Arad', 'Neamt')
    print("\n--- Resultado para (b) Arad -> Neamt ---")
    print(f"Melhor caminho encontrado: {' -> '.join(path_b)}")
    print(f"Distancia total: {dist_b} km")

def generate_pdf_report(code_content, output_content):
    """Gera um PDF com o código e o output."""
    pdf = FPDF()
    pdf.add_page()

    # Título Principal
    pdf.set_font("Courier", 'B', 16)
    pdf.cell(0, 10, 'Questao 1', ln=True, align='C')
    pdf.ln(10)

    # Seção do Código Fonte
    pdf.set_font("Courier", 'B', 12)
    pdf.cell(0, 10, 'Codigo Fonte:', ln=True)
    pdf.set_font("Courier", '', 8)
    # Usa multi_cell para texto de múltiplas linhas
    pdf.multi_cell(0, 5, code_content)

    # Adiciona uma nova página para o output para melhor formatação
    pdf.add_page()

    # Seção do Output
    pdf.set_font("Courier", 'B', 12)
    pdf.cell(0, 10, 'Output da Execucao:', ln=True)
    pdf.set_font("Courier", '', 10)
    pdf.multi_cell(0, 5, output_content)

    pdf_file_name = "resultado_questao_1.pdf"
    pdf.output(pdf_file_name)
    return pdf_file_name

# --- Bloco Principal de Execução ---
if __name__ == "__main__":
    # Captura o output da execução
    output_buffer = io.StringIO()
    with redirect_stdout(output_buffer):
        execute_and_capture_output()
    output_content = output_buffer.getvalue()

    # Lê o conteúdo do próprio script
    try:
        # __file__ é o caminho para o script atual
        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}"

    # Gera o PDF
```

```
try:
    pdf_file = generate_pdf_report(code_content, output_content)
    # Imprime uma mensagem final no console real
    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}")
    print("Verifique se a biblioteca 'fpdf2' esta instalada (pip install fpdf2).")
```

Output da Execucao:

```
--- Questao 1: Algoritmo Genetico para Caminho Mais Curto ---

Buscando melhor caminho de Oradea para Eforie...
Geracao 50: Melhor distancia = 698
Geracao 100: Melhor distancia = 698
Geracao 150: Melhor distancia = 698
Geracao 200: Melhor distancia = 698
Geracao 250: Melhor distancia = 698
Geracao 300: Melhor distancia = 698

--- Resultado para (a) Oradea -> Eforie ---
Melhor caminho encontrado: Oradea -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucareste ->
Urziceni -> Hirsova -> Eforie
Distancia total: 698 km

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

Buscando melhor caminho de Arad para Neamt...
Geracao 50: Melhor distancia = 824
Geracao 100: Melhor distancia = 824
Geracao 150: Melhor distancia = 824
Geracao 200: Melhor distancia = 824
Geracao 250: Melhor distancia = 824
Geracao 300: Melhor distancia = 824

--- Resultado para (b) Arad -> Neamt ---
Melhor caminho encontrado: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucareste ->
Urziceni -> Vaslui -> Iasi -> Neamt
Distancia total: 824 km
```