UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira
Teoría dos Grafos

Relatório do Projeto

Parte 1

Nome do Integrante	RA
Isabelle Ramos de Azevedo Ferreira	10365077
Júlia Rampani	10395600
Lucas Kenzo Kawamoto	10396359

Relatório

Título do projeto:

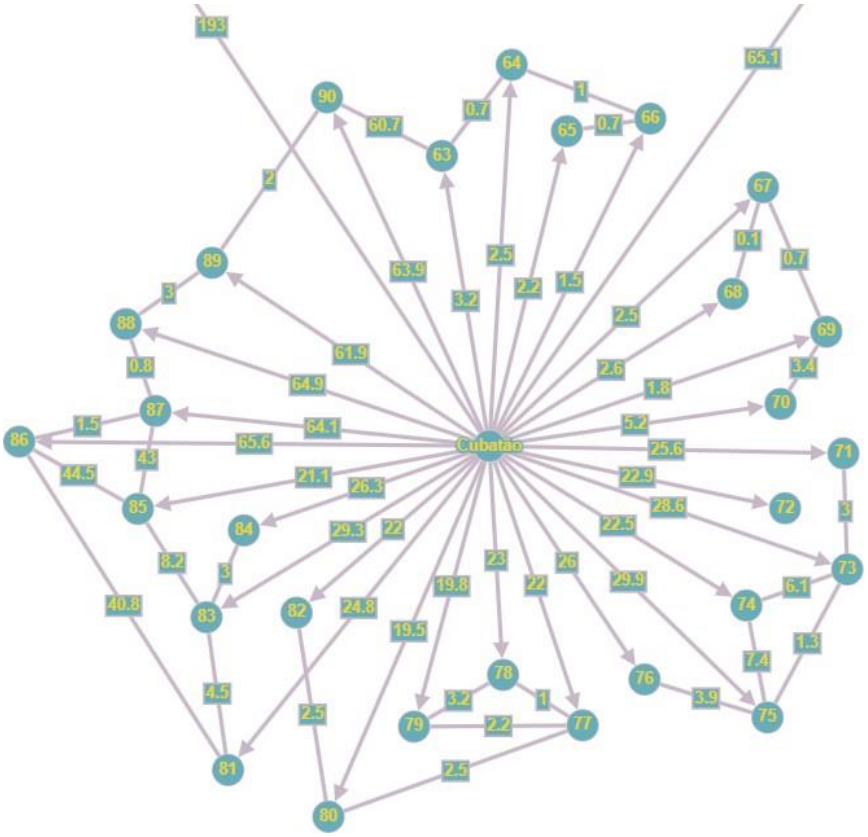
- Criação de grafo das distâncias entre as refinarias e postos do estado de SP.

Objetivo ODS:

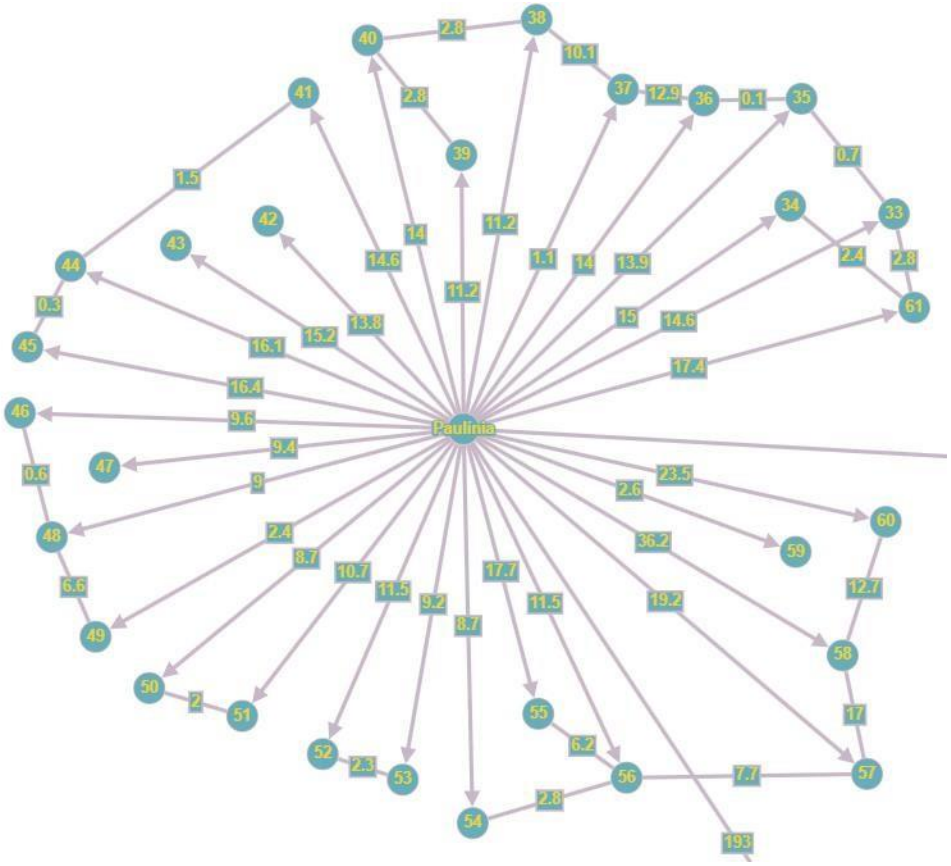
- Este projeto faz parte do Objetivo 7: Energia Limpa e Acessível: Analisar a distribuição geográfica das refinarias e postos de combustível pode ajudar a entender como o acesso a fontes de energia, como gasolina, diesel e etanol, está distribuído na cidade e se é acessível para todos os residentes.

Grafos criados:

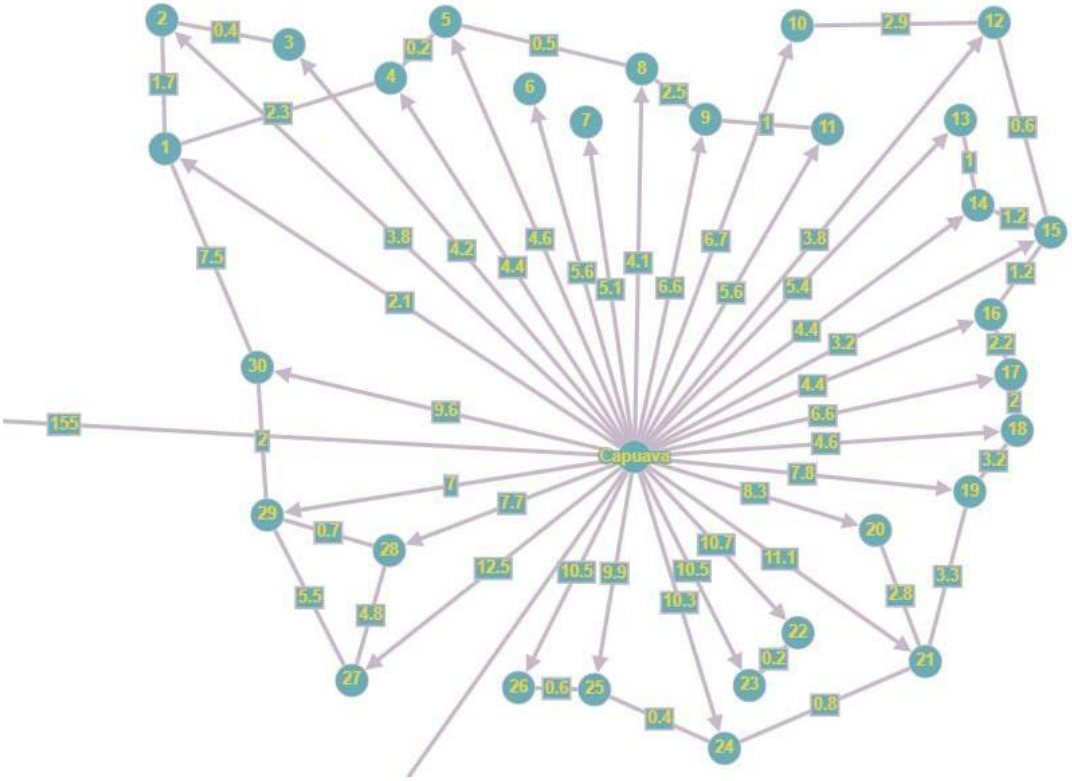
- Grafo de Cubatão:



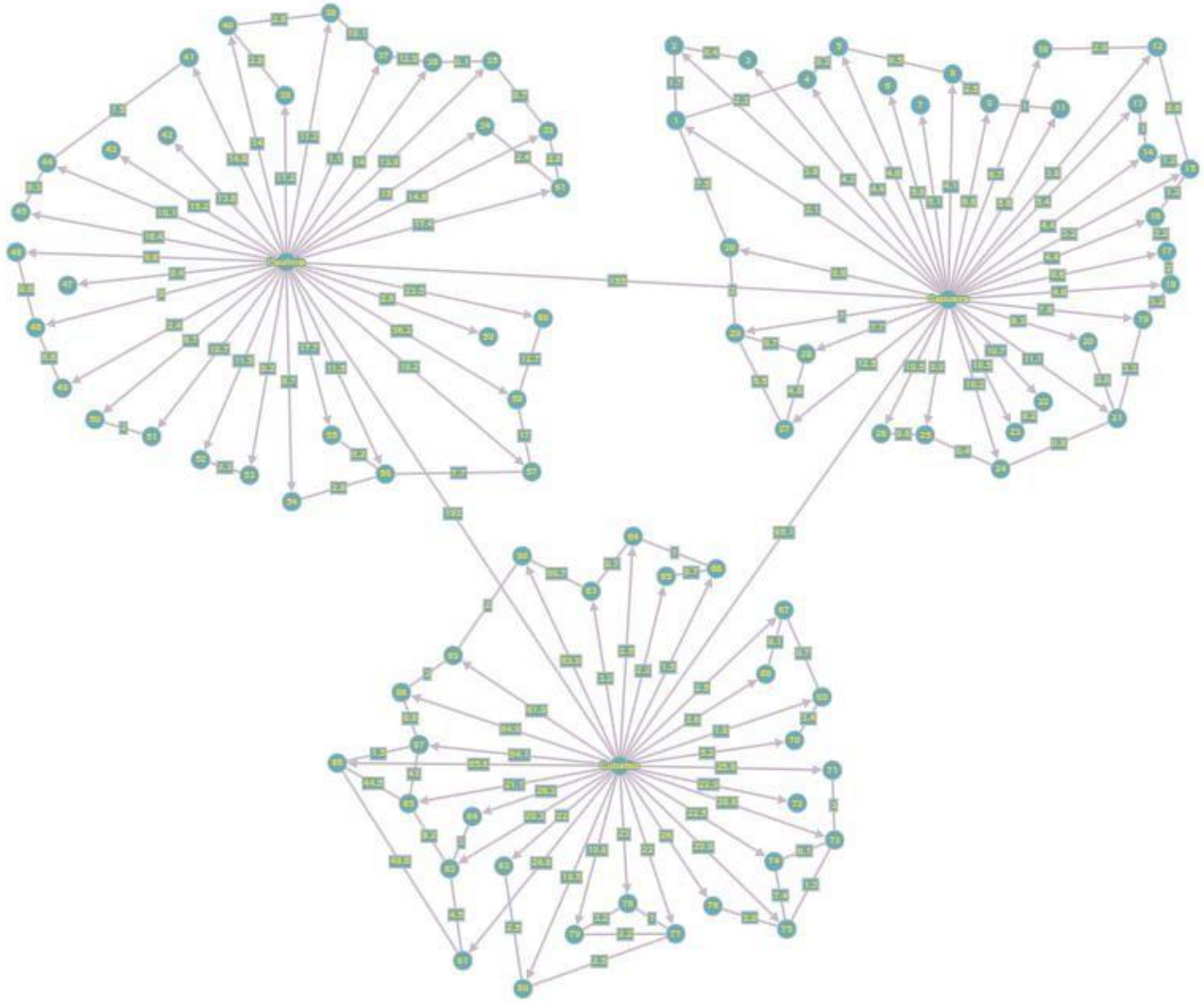
- Grafo de Paulínia:



- Grafo de Capuava:



- Grafo completo:



- **Obs: Para mais detalhes, acessar o link do grafo que se encontra no final do relatório.**

- **Obs2: A partir daqui, utilizaremos a denomização padronizada:**
  - **100 – Cubatão;**
  - **200 – Paulínia;**
  - **300 – Capuava.**

Conteúdo do arquivo grafo.txt:

100 64 2.5
100 65 2.2
100 66 1.5
100 67 2.5
100 68 2.6
100 69 1.8
100 70 5.2
100 71 25.6
100 72 22.9
100 73 28.6
100 74 22.5
100 75 29.9
100 76 26
100 77 22
100 78 23
100 79 19.8
100 80 19.5
100 81 24.8
100 82 22
100 83 29.3
100 84 26.3
100 85 21.1
100 86 65.6
100 87 64.1
100 88 64.9
100 89 61.9
100 90 63.9
100 300 193
100 200 65.1
63 64 0.7
64 66 1
65 66 0.7
67 68 0.1
67 69 0.7
69 70 3.4
71 73 3
73 74 6.1
73 75 1.3
74 75 7.4
75 76 3.9
77 78 1
77 79 2.2
78 79 3.2
77 80 2.5
80 82 2.5
81 83 4.5
81 86 40.8
83 84 3
83 85 8.2
85 86 44.5
85 87 43
86 87 1.5
87 88 0.8
88 89 3
89 90 2
90 63 60.7
200 1 2.1
200 2 3.8
200 3 4.2
200 4 4.4
200 5 4.6
200 6 5.6
200 7 5.1
200 8 4.1
200 9 6.6
200 10 6.7
200 11 5.6
200 12 3.8
200 13 5.4
200 14 4.4
200 15 3.2
200 16 4.4
200 17 6.6
200 18 4.6
200 19 7.8
200 20 8.3
200 21 11.1
200 22 10.7
200 23 10.5
200 24 10.3
200 25 9.9
200 26 10.5
200 27 12.5
200 28 7.7
200 29 7
200 30 9.6
200 300 155
200 100 65.1
1 2 1.7
1 4 2.3
2 3 0.4
4 5 0.2
5 8 0.5
8 9 2.5
9 11 1
10 12 2.9
12 15 0.6
13 14 1
14 15 1.2
15 16 1.2
16 17 2.2
17 18 2
18 19 3.2
19 21 3.3
20 21 2.8
22 23 0.2
21 24 0.8
24 25 0.4
25 26 0.6
27 28 4.8
28 29 0.7
29 30 2
30 1 7.5
300 33 14.6
300 34 15
300 35 13.9
300 36 14

300 37 1.1
300 38 11.2
300 39 11.2
300 40 14
300 41 14.6
300 42 13.8
300 43 15.2
300 44 16.1
300 45 16.4
300 46 9.6
300 47 9.4
300 48 9
300 49 2.4
300 50 8.7
300 51 10.7
300 52 11.5
300 53 9.2
300 54 8.7
300 55 17.7
300 56 11.5
300 57 19.2
300 58 36.2
300 59 2.6
300 60 23.5
300 61 17.4
300 200 155
300 100 193
33 35 0.7
35 36 0.1
36 37 12.9
37 38 10.1
38 40 2.8
39 40 2.8
41 44 1.5
44 45 0.3
46 48 0.6
48 49 6.6
50 51 2
52 53 2.3
54 56 2.8
55 56 6.2
56 57 7.7
57 58 17
58 60 12.7
61 34 2.4
61 33 2.8
63 200 100
30 63 100
30 300 100

Postos e refinarias utilizados:

Postos de gasolina perto da refinaria Cubatão

62 - Av. 31 de Março, 1259 - Nova Mirim, Praia Grande - SP, 11704-700
63 - Av. Ayrton Senna da Silva, 1016 - Sítio do Campo, Praia Grande - SP, 11320-200
64 - R. Moacyr de Camargo, 54 - Tupiry, Praia Grande - SP, 11719-050
65 - Av. Ayrton Senna da Silva, 608 - Sítio do Campo, Praia Grande - SP, 11724-210
66 - Av. Pres. Kennedy, 4854 - Vila Tupi, Praia Grande - SP, 11703-200
67 - Av. Pres. Kennedy, 10276 - Solemar, Praia Grande - SP, 11709-000
68 - Av. Guadalajara, 302 - Guilhermina, Praia Grande - SP, 11702-210
69 - Av. Dr. Roberto de Almeida Vinhas, 2860 - Vilamar, Praia Grande - SP, 11702-210
70 - Av. Pres. Kennedy, 1540 - Vila Caiçara, Praia Grande - SP, 11717-260
71 - Av. Ayrton Senna da Silva, 1511 - Praia Grande, SP, 11726
72 - Av. Ayrton Senna da Silva, 500 - Sítio do Campo, Praia Grande - SP, 11726-000
73 - Av. Nossa Sra. de Fátima, 651 - Vila Caiçara, Praia Grande - SP, 11706-300
74 - Av. 31 de Março, 611 - Nova Mirim, Praia Grande - SP, 11704-700
75 - R. Pernambuco, 618 - Boqueirão, Praia Grande - SP, 11700-010
76 - Av. 31 de Março, 1259 - Nova Mirim, Praia Grande - SP, 11704-700
77 - Av. Martins Fontes, 649 - Vila Nova, Cubatão - SP, 11520-110
78 - Av. Henry Borden, 1235 - Vila Santa Rosa, Cubatão - SP, 11515-000
79 - Av. 9 de Abril, 1068 - Vila Elizabeth, Cubatão - SP, 11505-000
80 - Av. Martins Fontes, 649 - Vila Nova, Cubatão - SP, 11525-090
81 - Av. Joaquim Miguel Couto, 1170 - Vila Couto, Cubatão - SP, 11510-010
82 - R. São Paulo, 295 - Jardim Sao Francisco, Cubatão - SP, 11500-020
83 - Av. Joaquim Miguel Couto, 619 - Vila Paulista, Cubatão - SP, 11510-010
84 - Av. 9 de Abril, 2916 - Centro, Cubatão - SP, 11525-090
85 - Av. dos Bandeirantes, 3686 - Parque Colonial, São Paulo - SP, 04553-003
86 - Av. Santo Amaro, 3240 - Brooklin, São Paulo - SP, 04556-200
87 - Av. Hélio Pellegrino, 1170 - Vila Nova Conceição, São Paulo - SP, 04513-100
88 - R. Vieira de Morais, 683 - Campo Belo, São Paulo - SP, 04817-011
89 - Av. Padre Antônio José dos Santos, 1575 - Cidade Monções, São Paulo - SP, 04563-000
90 - Av. Padre Antônio José dos Santos, 790 - Cidade Monções, São Paulo - SP, 04563-003

Postos de gasolina perto da refinaria Paulínia

1 - Tobras Distribuidora de Combustíveis - 14,6 km
2 - Cosan Combustíveis e Lubrificantes - 15 km
3 - Distribuidora de Combustíveis Torrão - 13,9 km
4 - Ipiranga Produtos de Petróleo Sa - 14 km
5 - Gol Combustíveis - 14 km
6 - Auto Posto Planalto - 1,1 km
7 - Royal FIC - Posto de Combustível - 11,2 km
8 - Via Petro Combustíveis Ltda - 11,2 km

9 - Masut Combustíveis - 14 km
10 - Taurus Distribuira de Petroleo Ltda - 14,6 km
11 - Monte Cabral - 13,8 km
12 - Raízen Combustíveis - Área 1 - 15,2 km
13 - Gran Petro Distribuidora de Combustível - 16,1 km
14 - Tercom Terminal Armazenagem Combustíveis - 16,4 km
15 - Rede Sol Fuel Distribuidora - 9,6 km
16 - Posto Petrobras - 9,4 km
17 - Ciapétro Distribuidora de Combustíveis - 9 km
18 - Torrão Distribuidora de Combustível - Base Norte - 2,4 km
19 - Posto Jardim Europa - 8,7 km
20 - Auto Posto Rodoshopping - 10,7 km
21 - Posto Cazellato BR - 11,5 km
22 - Auto Posto Novo Jardim de Paulínia - 9,2 km
23 - Quality Distribuidora de Combustíveis - 8,7 km
24 - Auto Posto Santa Carolina - 17,7 km
25 - Posto Cazellato Shell - 11,5 km
26 - Posto 7 Cherokee - 19,2 km
27 - Posto Piraju - 36,2 km
28 - Petronac Combustíveis - 2,6 km
29 - Posto Moraes - 23,5 km
30 - Auto Posto Sucão (Ipiranga) - 17,4 km

Postos de gasolina perto da refinaria Capuava

31 - Auto Posto Polo Petroquímico - 2,1 km
32 - Novo Ouro Negro - Posto de Combustível - 3,8 km
33 - Auto Posto Setee - 4,2 km
34 - Auto Posto Portal SMS LTDA - Shell - 4,4 km
35 - Auto Posto San Raphael - 4,6 km
36 - Posto Shell - 5,6 km
37 - Auto Posto 2 Amigos - 5,1 km
38 - Auto Posto Paladino - 4,1 km
39 - Auto Posto BR - Barbacena - 6,6 km
40 - Auto Posto Capricho - 6,7 km
41 - Auto Posto Artur de Queiros - 5,6 km
42 - Ale - 3,8 km
43 - Petrobras - 5,4 km
44 - Posto Kll - 4,4 km
45 - Posto Petrobras - 3,2 km
46 - Auto Posto ACP - 4,4 km
47 - Posto Papa João XXIII Petrobras - 6,6 km
48 - Tavarede Auto Posto - 4,6 km
49 - Auto Posto Shell San Pietro - 7,8 km
50 - Auto Posto Boxter - 8,3 km

51 - Posto Serviço EFR - BR - 11,1 km
52 - Auto Posto Jasmim - Rede EcoPonto - 10,7 km
53 - Rede Fevereiro - Camboriú - 10,5 km
54 - Carrefour Posto São Bernardo Vergueiro - 10,3 km
55 - POSTO SHELL - 9,9 km
56 - Posto Petrobras - Auto Posto Baeta Neves LTDA - 10,5 km
57 - Rede Fevereiro - Nova Petrópolis - 12,5 km
58 - Auto Posto Gastec - 7,7 km
59 - Posto Rede 1000 - 7 km
60 - Auto Posto Raio Dourado - 9,6 km

Programa Python Completo:

```python
def ler_grafo2(nome_arquivo):
    grafo = {}
    vertices = set()
    with open(nome_arquivo, 'r') as arquivo:
        linhas = arquivo.readlines()
        for linha_num, linha in enumerate(linhas, start=1):
            valores = linha.split()
            if len(valores) == 3:
                origem, destino, distancia = valores
                if origem not in grafo:
                    grafo[origem] = {}
                if destino not in grafo:
                    grafo[destino] = {}
                # Removendo "km" dos valores de distância e convertendo para float
                distancia = float(distancia)
                grafo[origem][destino] = distancia
                grafo[destino][origem] = distancia
                vertices.add(origem)
                vertices.add(destino)
            else:
                print(f"Erro na linha {linha_num}: A linha '{linha.strip()}' não possui o formato esperado (origem destino distância).")

    return grafo


# In[2]:


def gerar_matriz_adjacencia(grafo):
    vertices = sorted(grafo.keys())
    tamanho = len(vertices)
    matriz = [[0] * tamanho for _ in range(tamanho)]

    indice_vertices = {v: i for i, v in enumerate(vertices)}

    for origem in grafo:
        for destino, distancia in grafo[origem].items():
            matriz[indice_vertices[origem]][indice_vertices[destino]] = distancia

    return matriz
```

```python
def mostrar_matriz(matriz):
    for linha in matriz:
        print(" ".join(map(str, linha)))


# In[4]:


def dfs(grafo, vertice, visitados):
    stack = [vertice]
    while stack:
        v = stack.pop()
        if v not in visitados:
            visitados.add(v)
            for vizinho in grafo[v]:
                if vizinho not in visitados:
                    stack.append(vizinho)


# In[5]:


def verificar_conectividade(grafo):
    visitados = set()
    num_componentes = 0

    for vertice in grafo:
        if vertice not in visitados:
            dfs(grafo, vertice, visitados)
            num_componentes += 1

    if num_componentes == 1:
        return "C1", "O grafo é conexo."
    else:
        return "C0", f"O grafo possui {num_componentes} componentes desconexos."
```

```python
def conectar_componentes(grafo):
    visitados = set()
    componentes = []

    # Encontra os componentes conectados
    for vertice in grafo:
        if vertice not in visitados:
            componente = set()
            dfs(grafo, vertice, componente)
            componentes.append(componente)
            visitados.update(componente)

    # Conecta os componentes desconexos
    for i in range(len(componentes) - 1):
        componente_atual = componentes[i]
        proximo_componente = componentes[i + 1]
        v1 = next(iter(componente_atual))
        v2 = next(iter(proximo_componente))
        grafo[v1][v2] = 1.0
        grafo[v2][v1] = 1.0

    return grafo
```

```python
def criar_grafo_reduzido(grafo):
    grafo_reduzido = {}
    mapeamento = {}
    novo_vertice = 1

    for origem in grafo:
        if origem not in mapeamento:
            mapeamento[origem] = novo_vertice
            novo_vertice += 1

        for destino in grafo[origem]:
            if destino not in mapeamento:
                mapeamento[destino] = novo_vertice
                novo_vertice += 1

            vertice_origem = mapeamento[origem]
            vertice_destino = mapeamento[destino]

            if vertice_origem not in grafo_reduzido:
                grafo_reduzido[vertice_origem] = {}

            if vertice_destino not in grafo_reduzido[vertice_origem]:
                grafo_reduzido[vertice_origem][vertice_destino] = grafo[origem][destino]
            else:
                grafo_reduzido[vertice_origem][vertice_destino] += grafo[origem][destino]

    # Adicionar vértices que não estão conectados a nenhuma aresta
    for vertice in mapeamento.values():
        if vertice not in grafo_reduzido:
            grafo_reduzido[vertice] = {}

    # Garantir conexão entre Cubatão, Capuava e Paulínia
    if 100 in grafo_reduzido and 200 in grafo_reduzido:
        grafo_reduzido[100][200] = 65.1
        grafo_reduzido[200][100] = 65.1

    if 200 in grafo_reduzido and 300 in grafo_reduzido:
        grafo_reduzido[200][300] = 155
        grafo_reduzido[300][200] = 155
```

```
    if 100 in grafo_reduzido and 300 in grafo_reduzido:
        grafo_reduzido[100][300] = 193
        grafo_reduzido[300][100] = 193

    return grafo_reduzido


# In[8]:


def dijkstra(grafo, origem):
    if origem not in grafo:
        raise ValueError(f"O vértice {origem} não existe no grafo.")

    distancias = {vertice: float('inf') for vertice in grafo}
    predecessores = {vertice: None for vertice in grafo}
    distancias[origem] = 0
    visitados = set()
    nao_visitados = set(grafo.keys())

    while nao_visitados:
        menor_vertice = min(nao_visitados, key=lambda vertice: distancias[vertice])
        visitados.add(menor_vertice)
        nao_visitados.remove(menor_vertice)

        for vizinho, distancia in grafo[menor_vertice].items():
            if vizinho not in visitados:
                nova_distancia = distancias[menor_vertice] + distancia
                if nova_distancia < distancias[vizinho]:
                    distancias[vizinho] = nova_distancia
                    predecessores[vizinho] = menor_vertice

    return distancias, predecessores
```

```
def encontrar_caminho_minimo(origem, destino, predecessores):
    caminho = []
    atual = destino
    while atual != origem:
        caminho.insert(0, atual)
        atual = predecessores[atual]
    caminho.insert(0, origem)
    return caminho


# In[10]:


def menor_caminho(grafo, origem, destino):
    distancias, predecessores = dijkstra(grafo, origem)
    caminho_curto = encontrar_caminho_minimo(origem, destino, predecessores)

    if distancias[destino] == float('inf'):
        return float('inf'), []
    else:
        return distancias[destino], caminho_curto
```

```python
def colorir_vertices(grafo):
    cores = {}  # Dicionário para armazenar os vértices coloridos por cada cor
    cores_disponiveis = ['preto', 'amarelo', 'verde', 'roxo', 'vermelho']

    for vertice in grafo:
        vizinhos_coloridos = [cores.get(vizinho) for vizinho in grafo[vertice] if vizinho in cores]
        cor_disponivel = None

        for cor in cores_disponiveis:
            if cor not in vizinhos_coloridos:
                cor_disponivel = cor
                break

        if cor_disponivel is None:
            cor_disponivel = cores_disponiveis[0]

        cores[vertice] = cor_disponivel

    return cores


# In[12]:


def imprimir_cores(cores):
    cores_por_cor = {}
    for vertice, cor in cores.items():
        if cor not in cores_por_cor:
            cores_por_cor[cor] = []
        cores_por_cor[cor].append(vertice)

    print("\nVértices coloridos:")
    for cor, vertices in cores_por_cor.items():
        vertices_str = ", ".join(vertices)
        print(f"{cor} = {{{vertices_str}}}" if vertices else f"{cor} = {{}}")
```

```python
def calcular_graus(grafo):
    graus = {vertice: 0 for vertice in grafo}
    for vertice in grafo:
        graus[vertice] = len(grafo[vertice])
    return graus


# In[14]:


def imprimir_graus(graus):
    print("\nGraus dos vértices:")
    for vertice, grau in graus.items():
        print(f"Vértice {vertice}: Grau {grau}")


# In[15]:


def verificar_propriedades(grafo):
    euleriano = True
    hamiltoniano = True

    for vertice in grafo:
        if len(grafo[vertice]) % 2 != 0:
            euleriano = False

    grau_maximo = max(len(vizinhos) for vizinhos in grafo.values())
    if grau_maximo < len(grafo) - 1:
        hamiltoniano = False

    return euleriano, hamiltoniano
```

```python
def imprimir_propriedades(euleriano, hamiltoniano):
    if euleriano and hamiltoniano:
        print("O grafo é euleriano e hamiltoniano.")
    elif euleriano:
        print("O grafo é euleriano, mas não é hamiltoniano.")
    elif hamiltoniano:
        print("O grafo é hamiltoniano, mas não é euleriano.")
    else:
        print("O grafo não é euleriano nem hamiltoniano.")


# In[17]:


def mostrar_grafo2(grafo):
    for vertice, vizinhos in grafo.items():
        print(f"{vertice} -> {list(vizinhos.keys())}")


# In[18]:


def mostrar_grafo3(grafo):
    for vertice, vizinhos in grafo.items():
        vizinhos_formatados = [str(v) for v in vizinhos]
        print(f"{vertice} -> {', '.join(vizinhos_formatados)}")


# In[19]:


def mostrar_grafo4(grafo):
    for origem in grafo:
        for destino, peso in grafo[origem].items():
            print(f"{origem} -> {destino} : {peso}")
```

```python
def ler_grafo(arquivo):
    with open(arquivo, 'r') as f:
        conteudo = f.read().splitlines()
    grafo = {}
    for linha in conteudo:
        vertice, *arestas = linha.split()
        grafo[vertice] = arestas
    return grafo


# In[21]:


def gravar_grafo(arquivo, grafo):
    with open(arquivo, 'w') as f:
        for vertice, arestas in grafo.items():
            f.write(f"{vertice} {' '.join(arestas)}\n")


# In[22]:


def inserir_vertice(grafo, vertice):
    if vertice not in grafo:
        grafo[vertice] = []


# In[23]:


def inserir_aresta(grafo, inicio, fim):
    if inicio in grafo:
        grafo[inicio].append(fim)
```

```python
def remover_aresta(grafo, inicio, fim):
    if inicio in grafo:
        grafo[inicio].remove(fim)


# In[26]:


def mostrar_grafo(grafo):
    with open('grafo.txt', 'r') as f:
        for linha in f:
            print(linha.strip())  # strip() remove espaços em branco e quebras de linha extras


# In[27]:


def main():
    grafo = {}
    while True:
        print("\nMenu de opções:")
        print("a) Ler dados do arquivo grafo.txt")
        print("b) Gravar dados no arquivo grafo.txt")
        print("c) Inserir vértice")
        print("d) Inserir aresta")
        print("e) Remover vértice")
        print("f) Remover aresta")
        print("g) Mostrar conteúdo do arquivo")
        print("h) Mostrar grafo")
        print("i) Apresentar a conexidade do grafo")
        print("j) Encerrar a aplicação")
        print("1) Encontrar o menor caminho entre dois vértices")
        print("2) Coloração de vértices")
        print("3) Determinar grau dos vértices")
        print("4) Verificar se grafo é euleriano, hamiltoniano ou ambos")
        opcao = input("Escolha uma opção: ")
```

```
    if opcao == 'a':
        grafo = ler_grafo2('grafo.txt')
        print("Grafo lido do arquivo grafo.txt.")
    elif opcao == 'b':
        gravar_grafo('grafo.txt', grafo)
        print("Grafo gravado no arquivo grafo.txt.")
    elif opcao == 'c':
        vertice = input("Digite o vértice a ser inserido: ")
        inserir_vertice(grafo, vertice)
        print(f"Vértice {vertice} inserido com sucesso no grafo.")
    elif opcao == 'd':
        inicio = input("Digite o vértice de início da aresta: ")
        fim = input("Digite o vértice de fim da aresta: ")
        inserir_aresta(grafo, inicio, fim)
        print(f"Aresta de {inicio} para {fim} inserida com sucesso no grafo.")
    elif opcao == 'e':
        vertice = input("Digite o vértice a ser removido: ")
        remover_vertice(grafo, vertice)
        print(f"Vértice {vertice} removido com sucesso do grafo.")
    elif opcao == 'f':
        inicio = input("Digite o vértice de início da aresta a ser removida: ")
        fim = input("Digite o vértice de fim da aresta a ser removida: ")
        remover_aresta(grafo, inicio, fim)
        print(f"Aresta de {inicio} para {fim} removida com sucesso do grafo.")
    elif opcao == 'g':
        print("\nOs vértices que correspondem às refinarias são:")
        print("\n100 - Cubatão;")
        print("\n200 - Capuava;")
        print("\n300 - Paulínia")
        print('\n')
        mostrar_grafo(grafo)
    elif opcao == 'h':
        mostrar_grafo(grafo)
    elif opcao == 'i':
        nome_arquivo = "grafo.txt"
        grafo = ler_grafo2(nome_arquivo)
        categoria, comentario = verificar_conectividade(grafo)
        print(f"Categoria: {categoria}")
        print(f"Comentário: {comentario}")
```

```
            grafo_reduzido = criar_grafo_reduzido(grafo)
            print("\nOs vértices que correspondem às refinarias são:")
            print("\n100 - Cubatão;")
            print("\n200 - Capuava;")
            print("\n300 - Paulínia")
            print("\nGrafo Original:")
            mostrar_grafo4(grafo)
            print("\nGrafo Reduzido:")
            mostrar_grafo4(grafo_reduzido)
        elif opcao == 'j':
            print("Aplicação encerrada.")
            break
        elif opcao == '1':
            origem = input("Digite o vértice de origem: ")
            destino = input("Digite o vértice de destino: ")
            if origem in grafo and destino in grafo:
                distancia, caminho = menor_caminho(grafo, origem, destino)
                if distancia == float('infinity'):
                    print(f"Não há caminho entre {origem} e {destino}.")
                else:
                    print(f"Menor distância entre {origem} e {destino}: {distancia}")
                    print(f"Caminho: {' -> '.join(caminho)}")
            else:
                print("Um ou ambos os vértices não existem no grafo.")
        elif opcao == '2':
            cores = colorir_vertices(grafo)
            imprimir_cores(cores)
        elif opcao == '3':
            if grafo is None:
                print("Por favor, carregue um grafo primeiro.")
                continue
            graus = calcular_graus(grafo)
            imprimir_graus(graus)
        elif opcao == '4':
            if grafo is None:
                print("Carregue o grafo primeiro (opção 1).")
                continue

            graus = calcular_graus(grafo)
            imprimir_graus(graus)
```

```
            euleriano, hamiltoniano = verificar_propriedades(grafo)
            imprimir_propriedades(euleriano, hamiltoniano)
        else:
            print("Opção inválida!")


# In[28]:


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

Compilações do programa:

- Obs: Para o relatório não ficar muito grande, só colocamos fotos dos grafos mostrando as arestas conectadas ao vértice 100, que é Cubatão.

```
Menu de opções:
a) Ler dados do arquivo grafo.txt
b) Gravar dados no arquivo grafo.txt
c) Inserir vértice
d) Inserir aresta
e) Remover vértice
f) Remover aresta
g) Mostrar conteúdo do arquivo
h) Mostrar grafo
i) Apresentar a conexidade do grafo
j) Encerrar a aplicação
1) Encontrar o menor caminho entre dois vértices
2) Coloração de vértices
3) Determinar grau dos vértices
4) Verificar se grafo é euleriano, hamiltoniano ou ambos
Escolha uma opção: a
Grafo lido do arquivo grafo.txt.
```

```
Os vértices que correspondem às refinarias são:

100 - Cubatão;

200 - Capuava;

300 - Paulínia


100 64 2.5
100 65 2.2
100 66 1.5
100 67 2.5
100 68 2.6
100 69 1.8
100 70 5.2
100 71 25.6
100 72 22.9
100 73 28.6
100 74 22.5
100 75 29.9
```

```
100 76 26
100 77 22
100 78 23
100 79 19.8
100 80 19.5
100 81 24.8
100 82 22
100 83 29.3
100 84 26.3
100 85 21.1
100 86 65.6
100 87 64.1
100 88 64.9
100 89 61.9
100 90 63.9
100 300 193
100 200 65.1
```

```
Escolha uma opção: i
Categoria: C1
Comentário: O grafo é conexo.

Os vértices que correspondem às refinarias são:

100 - Cubatão;

200 - Capuava;

300 - Paulínia
```

```
Grafo Original:
100 -> 64 : 2.5
100 -> 65 : 2.2
100 -> 66 : 1.5
100 -> 67 : 2.5
100 -> 68 : 2.6
100 -> 69 : 1.8
100 -> 70 : 5.2
100 -> 71 : 25.6
100 -> 72 : 22.9
100 -> 73 : 28.6
100 -> 74 : 22.5
100 -> 75 : 29.9
100 -> 76 : 26.0
100 -> 77 : 22.0
100 -> 78 : 23.0
100 -> 79 : 19.8
100 -> 80 : 19.5
100 -> 81 : 24.8
100 -> 82 : 22.0
100 -> 83 : 29.3
```

```
100 -> 84 : 26.3
100 -> 85 : 21.1
100 -> 86 : 65.6
100 -> 87 : 64.1
100 -> 88 : 64.9
100 -> 89 : 61.9
100 -> 90 : 63.9
100 -> 300 : 193.0
100 -> 200 : 65.1
```

```
Grafo Reduzido:
1 -> 2 : 2.5
1 -> 3 : 2.2
1 -> 4 : 1.5
1 -> 5 : 2.5
1 -> 6 : 2.6
1 -> 7 : 1.8
1 -> 8 : 5.2
1 -> 9 : 25.6
1 -> 10 : 22.9
1 -> 11 : 28.6
1 -> 12 : 22.5
1 -> 13 : 29.9
1 -> 14 : 26.0
1 -> 15 : 22.0
1 -> 16 : 23.0
1 -> 17 : 19.8
1 -> 18 : 19.5
1 -> 19 : 24.8
1 -> 20 : 22.0
```

```
1 -> 21 : 29.3
1 -> 22 : 26.3
1 -> 23 : 21.1
1 -> 24 : 65.6
1 -> 25 : 64.1
1 -> 26 : 64.9
1 -> 27 : 61.9
1 -> 28 : 63.9
1 -> 29 : 193.0
1 -> 30 : 65.1
```

```
1) Encontrar o menor caminho entre dois vértices
2) Coloração de vértices
3) Determinar grau dos vértices
4) Verificar se grafo é euleriano, hamiltoniano ou ambos
Escolha uma opção: 1
Digite o vértice de origem: 100
Digite o vértice de destino: 90
Menor distância entre 100 e 90: 63.9
Caminho: 100 -> 90
```

```
Escolha uma opção: 2

Vértices coloridos:
preto = {100, 63, 1, 3, 5, 6, 7, 9, 10, 13, 15, 17, 19, 20, 22, 24, 26, 27, 29, 33, 34, 36, 38, 39, 41, 42, 43, 45, 46, 47, 49, 50, 52, 54, 55, 57, 59, 60}
amarelo = {64, 65, 67, 70, 71, 72, 74, 76, 77, 81, 82, 84, 85, 88, 90, 300, 2, 4, 8, 11, 12, 14, 16, 18, 21, 23, 25, 28}
verde = {66, 68, 69, 73, 78, 80, 83, 86, 89, 200, 35, 37, 40, 44, 48, 51, 53, 56, 58, 61}
roxo = {75, 79, 87, 30}
```

```
Escolha uma opção: 3

Graus dos vértices:
Vértice 100: Grau 29
Vértice 64: Grau 3
Vértice 65: Grau 2
Vértice 66: Grau 3
Vértice 67: Grau 3
Vértice 68: Grau 2
Vértice 69: Grau 3
Vértice 70: Grau 2
Vértice 71: Grau 2
Vértice 72: Grau 1
Vértice 73: Grau 4
Vértice 74: Grau 3
Vértice 75: Grau 4
Vértice 76: Grau 2
Vértice 77: Grau 4
Vértice 78: Grau 3
Vértice 79: Grau 3
Vértice 80: Grau 3
Vértice 81: Grau 3
Vértice 82: Grau 2
Vértice 83: Grau 4
Vértice 84: Grau 2
Vértice 85: Grau 4
Vértice 86: Grau 4
Vértice 87: Grau 4
Vértice 88: Grau 3
Vértice 89: Grau 3
Vértice 90: Grau 3
Vértice 300: Grau 32
Vértice 200: Grau 33
Vértice 63: Grau 4
Vértice 1: Grau 4
Vértice 2: Grau 3
Vértice 3: Grau 2
Vértice 4: Grau 3
Vértice 5: Grau 3
Vértice 6: Grau 1
Vértice 7: Grau 1
```

```
4) Verificar se grafo é euleriano, hamiltoniano ou ambos
Escolha uma opção: 4
O grafo não é euleriano nem hamiltoniano.
```

Link do Github:

- **Programa completo:** https://github.com/LUCASBR8/GRAFOS/blob/main/grafo1%20(1).py
- **Arquivo de texto:** https://github.com/LUCASBR8/GRAFOS/blob/main/grafo.txt