Nome: Tiago Faustino de Siqueira **Matrícula**: 22102193

Exercício 1: Implementei uma classe Grafo com dois atributos: 1 dicionário para arestas e outro para objetos Vértice.Escolhi usar dicionários por 3 principais motivos: melhor desempenho, fácil acesso e fácil implementação. Cada Vértice é uma instância de classe que contém os atributos relevantes na aplicação dos algoritmos: rótulo (*str*), vizinhos (*set*), grau (*int*) e visitado (*bool*). Adicionalmente, uma classe Manager foi criada para processar arquivos de entrada e gerar o grafo correspondente, com um atributo temPeso para otimizar a carga de dados conforme a relevância do peso das arestas nos problemas.

Exercício 2: Utilizei um dicionário para agrupar vértices por nível e uma fila (lista) para registrar vértices já visitados. A escolha do dicionário para agrupar os vértices por nível é justificada pela capacidade de acesso rápido aos grupos de vértices correspondentes a cada nível e pela flexibilidade na manipulação de chaves dinâmicas. Já a escolha de uma lista para a fila é adequada para representar a natureza sequencial e ordenada da BFS, onde o primeiro vértice a ser adicionado é o primeiro a ser explorado, seguindo a lógica FIFO (First In, First Out).

Exercício 3: Eu separei o algoritmo descrito na apostila em mais partes e adaptei para a maneira que eu armazeno os componentes do meu grafo. Então vou listar as estruturas por método. No método de verificar se o grafo é conectado, eu utilizo um *set* para armazenar os vértices visitados. No método de encontrar um circuito euleriano eu uso duas listas, uma para armazenar a stack atual e outra para armazenar o circuito. Usar um set para armazenar vértices visitados permite um acesso rápido para verificar se um vértice já foi explorado ou não, o que é crucial nesse algoritmo para evitar repetições. As duas listas facilitam a gestão do algoritmo em tempo de execução, pois são ideais para operações de adição e remoção no fim, comportando-se como uma pilha (stack) que é fundamental nesse tipo de algoritmo de circuito fechado.

Exercício 4: Minha tentativa inicial de usar uma heap de Fibonacci falhou, levando à adoção de uma heap queue convencional para armazenar os vértices restantes. Esta escolha visou otimizar a busca pelo vértice com menor distância pendente (`arg min`), que aumentava muito a complexidade computacional. Além disso, usei 3 dicionários para armazenar distâncias em relação ao vértice escolhido (*D*), árvore de predecessores (*A*) e vértices já visitados (*C*). Os três dicionários utilizados permitem um gerenciamento eficiente de dados associados a cada vértice

Exercício 5: Optei por uma estrutura simples, usando uma lista para armazenar vértices, um dicionário (*D*) para matrizes associadas e um dicionário (*D_anterior*) para matrizes associadas, conforme o algoritmo da apostila. Usar listas para vértices ajudou a manter uma ordem consistente para referência cruzada com os índices das matrizes nos dicionários. Os dicionários são escolhidos para armazenar as matrizes devido à sua flexibilidade na atualização de valores e na capacidade de manipular grandes quantidades de dados associativos de forma eficiente, o que é ideal nesse algoritmo, pois ele requer múltiplas passagens para atualização incremental.