**Grafos - Relatório Atividade 1**

Nomes: Gabriel Turatti, Gabriela Furtado, Victor Gouvêa

**Questão 1:**

Foi criada uma classe para o Nodo do grafo, e uma classe para o grafo em si. Na classe do Nodo, foram usadas as propriedades:

Id, para representar o nodo  
rótulo, para representar o nome do nodo  
connections, para representar a quantidade de conexões que aquele nodo possui (por razões de tornar as outras funções do grafo mais eficientes no tempo de execução)   
vizinhos, para representar o dicionário que guarda a informação de que nodos são vizinhos e qual o peso da conexão deles.

Na classe do Grafo, foram utilizadas as propriedades:

V, como o dicionário de nodos presentes no grafo, sendo a posição do dicionário o id do nodo, e o valor o objeto do nodo em si  
nV, a quantidade de nodos (novamente, com o propósito de agilizar o funcionamento de funções que dependam do tamanho do grafo)  
E, como a lista de arestas do grafo  
nE, como a quantidade de arestas

Para os métodos que eram necessários programar, qtdVertices e qtdArestas utilizam do nV e nE, sendo dessa forma O(1), grau retorna o connections do nodo, rotulo retorna o rótulo do nodo, vizinhos retorna o dicionário de vizinhos do nodo, haAresta checa se o peso da conexão entre os dois nodos é menor que infinito, se for, é porque há uma aresta e ele retorna verdadeiro, peso simplesmente retorna o que tem na posição id do dicionário, que é o valor do peso da aresta.

A função de ler o arquivo, primeiramente irá escanear cada linha do arquivo em busca do caractere de ‘\*’, quando encontrar ele começara a ler as próximas linhas e criar nodos para cada uma delas, assim que encontrar outro caractere ‘\*’, ele começará a registrar as informações como arestas.

**Questão 3:**

Foi implementada uma versão do algoritmo de Hierholzer para encontrar Ciclos Eulerianos, seguindo o que foi visto em aula. Para indicar se o vértice é conhecido, foi utilizado um dicionário, cujas chaves são conjuntos (frozensets) contendo pares de vértices (já que são consideradas arestas, ou seja, a ordem não importa). O algoritmo principal realiza uma chamada a “buscaCiclo”, passando um vértice qualquer que tenha conexão, e nessa função o ciclo é salvo como uma lista, na qual os elementos são atributos “id” de objetos da classe “Node” (vértices). Tenta-se montar um ciclo a partir dos vizinhos do vértice com verificações de visita realizadas a partir do dicionário também passado como argumento, e caso haja algum subciclo, os novos elementos são inseridos na lista “ciclo”.

Com o retorno obtido de “buscaCiclo”, o algoritmo principal também verifica se todas as arestas foram visitadas, buscando algum valor falso no dicionário de visitas, e realiza a impressão na tela da saída esperada.

**Questão 5:**

A função do algoritmo Floyd-Warshall foi implementada como visto na aula. Primeiramente é criado uma matriz, e preenchida a informação dela com infinitos, representando a distância de cada conexão dos nodos, menos a distância de um nodo para si mesmo, que é sempre registrado como 0. Após isso, a função irá ler a lista de arestas, e atualizar os valores de infinito de acordo.

Durante o principal looping do algoritmo, pela forma como o for se comporta com dicionários no Python, foi necessário resgatar o valor dos indexes de cada vértice para depois ser feito a comparação de distância.