INE5413 - Grafos - Relatório da atividade 1

David Grunheidt Vilela Ordine - 16202253

Exercício 1: Para a criação do grafo, foi utilizado uma classe **'NotDirectedGraph'**. Essa classe possui três atributos. Primeiramente, o atributo '**vertices'** é definido como um dicionário, ou seja, um array associativo, e representam a associação entre um id de um vértice e a string representando seu nome. Já o atributo **'graph'** é também um dicionário que relaciona um id a um objeto do tipo Vértice, que contém todas as informações sobre tal vértice. Por fim, para garantir a operação de '**qtdArestas()'** seja feita em O(1), foi adicionado o atributo **'numberOfEdges'**, o qual controla o número de arestas conforme são adicionadas ou removidas.

Já a classe **'Vertex'** representa um vértice, contendo 5 atributos. Primeiramente, temos os atributos **'vertex_id'** e **'vertex_name'** do tipo string, identificando o vértice. Na sequência o atributo **'neighbors'** é um set que guarda todos os ids dos vértices vizinhos a este. Isso foi feito para que a operação **vizinhos(v)** seja feita em 0(1). O próximo atributo, **'edges'**, é um dicionário, onde a chave é um id de um vértice vizinho e o valor é o peso daquela aresta. Por fim, **'degree'** guarda o grau do vértice, para que a operação **grau(v)** seja feita em O(1).

Exercício 2: Para o algoritmo de busca em largura, foi escolhido um set para a estrutura de dados que guardaria os vértices visitados. Essa é uma boa escolha pois essa estrutura não permite adicionar um elemento caso ele já exista no conjunto, o que possibilita não verificar se tal elemento pertence ao conjunto. Já para a fila, foi utilizado uma lista, a qual possui o método pop(), responsável por remover o último elemento dessa lista, facilitando a implementação.

Exercício 3: Para o algoritmo de ciclo euleriano, utilizou-se o algoritmo de busca em largura anterior para verificar o tipo de grafo, assim como para verificar se uma certa aresta é uma ponte. Também foi utilizado a função que retorna os vizinhos de um certo vértice para realizar as iterações de próximos vértices válidos. Por fim, foi utilizado uma lista para calcular, recursivamente, qual o caminho do ciclo. Portanto, as única estruturas de dados usadas foram um set, retornada pelo método de busca em largura e pelo método de recuperar os vizinhos, e uma lista.

Exercício 4: Foi feito a implementação do algoritmo de Dijkstra. A estrutura de dados mais importante deste exercício é o **'dists'**, um dicionario de dicionarios. Nessa variável, uma chave é associada a um dicionário, o qual possui as seguintes chaves: **'min_distance'**, ou seja, a distância mínima calculada iterativamente ao longo da execução, **'parent'**, usado para calcular o caminho do vértice raiz até o vértice atual, e **'in_queue'**, usado para controlar a iteração de cálculo da distância mínima. Ao final, através deste dicionário, é possível calcular reversamente o caminho do vértice atual até o vértice raiz.

Exercício 5: Utilizou-se a lógica do exercício anterior para recuperar as distâncias mínimas de um certo vértice inicial para todos os outros vértices. Assim, iterou-se sobre todos os vértices do grafo, e para cada vértice aplicou-se o método implementado anteriormente. Assim, calculou-se todas as distâncias mínimas de todos os vértices para todos os vértices. Portanto, nenhuma estrutura de dados foi usada especificamente para esse exercício.