Relatório — Atividade 1

Nome: Lucas Pagotto Coutinho de Oliveira(18201971)
Bruno Pamplona Huebes (19102917)
Albano Antônio Mendes(18150742)

Matéria/Turma:INE5413-04208

Questão 1) Representação

Implementei uma classe para representar grafos não-direcionados, sejam eles ponderados ou não. Os atributos da classe são: número de vértices, número de arestas, boolean que sinaliza se o grafo é ponderado, lista de vértices e matriz de arestas (ou de adjacências).

A lista de vértices armazena nas posições i − 1 os rótulos de cada vértice v i . A matriz de arestas é uma matriz triangular inferior sem diagonal principal, visto que essa classe representa apenas grafos não-direcionados. Devido a esta otimização de memória, cada aresta {i, j} é armazenada nas posições (max(i, j) − 2, min(i, j) − 1) da matriz. Os elementos dessa matriz são 0's e 1's se o grafo for não-ponderado, sinalizando a existência das arestas. Caso o grafo seja ponderado, os elementos da matriz são os pesos das arestas, sendo infinito para arestas não existentes.

A classe contém todas as funções requeridas pelo enunciado. Vale mencionar que a função que lê arquivos de grafo recebe o nome dos arquivos pela execução atráves do prompt de comando. Os arquivos de grafo a serem lidos devem estar contidos na pasta networks.

Questão 2) Buscas

O programa recebe pelo prompt de comando o nome do arquivo de grafo a ser lido e o índice do vértice de origem, e então instancia um grafo não-ponderado. (Exemplo de execução: python busca_em_largura.py facebook_santiago.net 99)

São utilizadas três listas: de vértices já conhecidos, de distâncias até cada vértice e dos antecessores que levaram até cada vértice. Também é utilizada uma fila para armazenar as visitas durante a busca em largura. O algoritmo de busca em largura
foi reproduzido tal como consta na p. 26 das Anotações da Disciplina. Por fim, foi utilizado um dicionário para indexar os vértices que foram percorridos em cada nível da busca, mostrando-os em ordem ascendente de níveis.

Questão 3) Ciclo Euleriano

Novamente, pelo prompt de comando recebe-se o nome do arquivo de grafo a ser lido (como por exemplo: python3 hier.py ContemCicloEuleriano.net). Utiliza-se uma versão do algoritmo de Hierholzer (não especificamente na mesma lógica das Anotações da Disciplina). Como referência para a versão do algoritmo utilizada, seguiu-se o site de computação Slay Study (Slay Study; https://slaystudy.com/hierholzers-algorithm/, visitado em 29 d maio de 2022). Note que essa implementação, em geral, não será compatível com grafos valorados.

Utiliza-se uma lista de listas para armazenar as arestas, junto de um booleano para representar se já foi visitado. Ainda, há listas para o ciclo euleriano e o subciclo gerado para cada vértice visitado. A ideia é adcionar valores do subciclo para o ciclo, a partir da estrutura que armazena as arestas, quando um vértice não tiver mais ligações não visitadas. Esse algoritmo tem, aproximadamente, um crescimento de O(|E|), com |E| sendo o número de arestas.

Questão 4) Algoritmo de Bellman-Ford

O programa recebe pelo prompt de comando o nome do arquivo de grafo a ser lido e o índice do vértice de origem, e então instancia um grafo ponderado. (Exemplo de execução: python bellman_ford.py fln_pequena.net 1)

São utilizadas três listas: de vértices já conhecidos, de distâncias até cada vértice e dos antecessores que levaram até cada vértice. O algoritmo de Bellman-Ford foi reproduzido tal como consta na p. 50 das Anotações da Disciplina. Por fim,percorreram-se todos os vértices do grafo, reconstruindo os caminhos mínimos através da lista de antecessores e mostrando-os juntamente com suas distâncias.

Questão 5) Algoritmo de Floyd-Warshall

A implementação segue o algoritmo na página 57 das Anotações da Disciplina. Pode ser executado com o prompt: python3 floydwarshall.py fln_pequena.net (a última variável pode ser substituída com o arquivo de grafo de preferência).

O algoritmo gera uma uma matriz de relação entre pares de vértices U e V, e atualiza ela N vezes (para o número de vértices no grafo), ou seja, para usar cada vértice como meio de ligação no caminho, mantendo apenas o caminho mínimo encontrado. Assim, tem-se ordem de crescimento O(|V|^3).