Universidade Federal De Viçosa (UFV) - Campus Florestal
Disciplina: Teoria e Modelo de Grafos (CCF 331)
Professor(a): Marcus Henrique Soares Mendes

Trabalho prático 1

Biblioteca para manipulação de Grafos

Autores:

Miguel Antônio Ribeiro e Silva - 4680
João Victor Graciano Belfort de Andrade - 4694
Mateus Henrique Vieira Figueiredo - 4707
Alan Gabriel Martins Silva - 4663

Introdução

O trabalho tem como principal problema a construção de uma biblioteca para manipulação de grafos não orientados ponderados.

Os grafos são úteis na representação de problemas da vida real.

Podem ser cidades, redes de estradas ou redes de computadores. Até mesmo os movimentos de um cavalo num tabuleiro de xadrez podem ser representados através de um grafo.

E depois de representá-los corretamente, o que podemos descobrir? O caminho mais curto entre duas cidades num mapa; dadas as coordenadas de *n* cidades, que estradas construir de modo que o número de quilômetros de estrada seja mínimo mas fiquem todas conectadas; dado um mapa de uma casa (em que paredes e chão são representados com caracteres diferentes) saber qual a divisão com maior área; entre outros. [0]

As possibilidades são grandes.

Por isso, a biblioteca apresentada a seguir auxiliará nas complexas buscas e análises dessa estrutura de dados, desde as informações básicas até a busca pelo caminho ótimo, usando algoritmos consolidados na área da computação.

Metodologia

Após a formação do grupo, destacamos as principais tarefas a serem cumpridas para a realização do trabalho prático. Como:

- Implementação da entrada por arquivos de texto
- Desenvolvimento de funções simples
- Pesquisas e estudos sobre as características de um grafo não orientado, ponderado
- Busca por algoritmos de menor caminho
- **Função extra** para geração de arquivos teste
- Funções de conversão de arquivos (.Json para .txt e .txt para .Json)
- Elaboração da documentação e vídeo

O código fonte foi desenvolvido em **Python 3** [1] e versionado no **GitHub**[2]**,** visando que seria a melhor forma de compartilhamento do mesmo entre os integrantes do grupo.
Para uma melhor organização e visualização do projeto, este foi dividido em subpastas.

> **-./functions** - implementação da biblioteca para análise dos grafos e conversão de arquivos
> **-./json-files** - arquivos .json utilizados
> **-./txt-files** - arquivos .txt utilizados

Detalhes Técnicos de Implementação

As funções foram implementadas no arquivo ./functions/weighted-graph.py e ./functions/converter.py, referenciadas: [4][5][6][7]

Biblioteca weighted-graph.py

Possui duas classes, **class Edge**, para representação das arestas do grafo, e a **class GraphWeighted**, usada para representar um grafo não direcionado, ponderado, usando lista de adjacências

Class Edge

Atributos:

connected vertex: (int) um inteiro que representa um vértice.
weight: (float) o peso da aresta.

Métodos:

_str__(self): retorna uma representação de string da aresta.

Class GraphWeighted

Atributos:

v number: (int) um inteiro que representa o número de vértices.
adj : (dict) um dicionário que representa a lista de adjacências do grafo.

Métodos:

get_vertex_sequence(self): retorna a sequência dos vértices.
get_first_vertex(self): retorna o primeiro vértice do grafo.
add_edge(self, a, b, w)**:** adiciona uma nova aresta ao gráfico.
get_order(self)**:** retorna a ordem do grafo.
size(self): retorna o tamanho do grafo.
get_neighbours(self, v): retorna os vizinhos de um vértice.
degree_of_vertex(self, v): retorna o grau de um vértice.

degree sequence(self): retorna a sequência de graus do grafo.

eccentricity(self, v): retorna a excentricidade de um vértice.

bellman_ford(self, vertex): retorna as distâncias de todos os vértices para um vértice usando o algoritmo de Bellman-Ford.

radius(self): retorna o raio do grafo.

diameter(self)*:* retorna o diâmetro do grafo.

center(self): retorna o centro do grafo.

dfs(self, v): Retorna a dfs (busca em profundidade) do grafo.

dfs_util(self, v, visited)*:* Função auxiliar para dfs(self,v).

dfs not visited(self): retorna vértices que não foram visitados pela dfs.

closeness_centrality(self, v): retorna a centralidade de proximidade de um vértice.

minimum_path(self, v, w)*:* retorna o caminho mínimo entre dois vértices.

Biblioteca converter.py

Não possui classes, apenas métodos. Foi utilizado **import json** como auxílio.

Métodos:

vertex_list(text_file): retorna uma lista com os vértices de um grafo, pelo arquivo.

j*son_to_text(json_file, text_file):* converte um arquivo .json (gerado em [3]) em um arquivo.txt.

def text_to_json(text_file, json_file): converte um arquivo .txt em um arquivo .json (para utilizar em [3])

main.py

Ponto de partida para a execução do projeto, aqui **todos os métodos** podem ser devidamente testados.

Primeiramente, o usuário deve digitar o diretório de um arquivo .txt, compactado de acordo com a especificação do trabalho ou gerado pelo nosso script (ver abaixo).

O programa retornará todas as características do grafo, explicadas acima, além disso, antes de encerrar, o programa perguntará se o usuário quer converter algum arquivo para um outro formato.

generate-graph.py

Script extra implementado; usado para gerar um arquivo de texto, compactado, randomizado.

O usuário deverá digitar a quantidade de vértices (inteiro positivo) que deseja e a probabilidade de um vértice estar ligado em outro. Os pesos são sempre positivos.

Será gerado um arquivo de texto, na pasta txt-files.

Considerações finais

Após o fim do trabalho prático podemos apontar certas dificuldades e facilidades encaradas pelo grupo durante a implementação desse projeto. Notamos uma certa facilidade na hora de entender como um grafo funciona e como poderíamos implementar as funções mais intuitivas.

Tivemos dificuldades de implementar os algoritmos de caminho mínimo (**Bellman-Ford**) e excentricidade, visto que o grafo não é direcionado e pode possuir pesos negativos, que quase sempre encontra um ciclo negativo.

A conversão de arquivos foi demorada e pouco intuitiva de criar, visto que o site [3] possuía poucas informações de implementação.

Contudo é visível o proveito e as lições aprendidas durante esse período de desenvolvimento do trabalho prático.

Referências

[0]: https://www.revista-programar.info/artigos/grafos-1a-parte/

[1]: https://www.python.org/

[2]: https://github.com/

[3]: https://paad-grafos.herokuapp.com/

[4]: https://www.geeksforgeeks.org/graph-and-its-representations

[5]: https://www.geeksforgeeks.org/building-an-undirected-graph-and-finding-shortest-path-using-dictionaries-in-python/

[6]: https://www.educative.io/answers/how-to-implement-a-graph-in-python

[7]: https://www.lavivienpost.net/weighted-graph-as-adjacency-list