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

Trabalho prático 2
Mais funções para a biblioteca de 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 continuar a implementação da biblioteca de manipulação de grafos.

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.

As funções a seguir foram implementadas em uma biblioteca para manipulação dos mesmos.

Metodologia

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

- Implementação das funções mais simples.
- Estudos sobre os algoritmos de Árvore Geradora Mínima e Matching Máximo.
- Ler os slides.
- 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 testadas na main2.py

Biblioteca weighted-graph.py

Métodos novos adicionados:

QUESTÃO 01

has_cycle(self): retorna True or False se o grafo tem ciclo.
has_cycle_util(self,v,parent,visited): função auxiliar para a has_cycle(self).
paramêtros*: v - vértice.*
parent - parente do vértice v.
visited - dicionário com os vértices visitados.

QUESTÃO 02

vertex_list(self): retorna a lista de vértices.
minimum_vertex_cover_heuristic(self): retorna a cobertura mínima de vértices usando o algoritmo heurístico apresentado em sala de aula.

QUESTÃO 03

edge_weight(self, v, w): retorna o peso da aresta entre dois vértices v e w.
minimum_spanning_tree(self, s): retorna a árvore geradora mínima inserida em um arquivo *s*MST.txt, na pasta ./txt-files, usando o **algoritmo de Prim**, apresentado em sala de aula.
parâmetros: s - nome do arquivo que será criado.

QUESTÃO 04

maximum_matching(self): heurística que calcula um matching máximo de um vértice bipartido.

main2.py

Ponto de partida para a execução do projeto, aqui **todos os métodos novos criados** 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 as características do grafo, explicadas acima.

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 poderíamos implementar as funções mais intuitivas.

Tivemos dificuldades de implementar os algoritmos de árvora geradora mínima (**Prim**) e matching máximo.

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]: Slides do professor.