Relatório:
Biblioteca de Grafo + Estudos de Caso

Instituto Federal de Brasília
Disciplina: Teoria dos Grafos
Professor: Raimundo Vasconcelos
Aluno(a): Cinthia Mie Nagahama Ungefehr e Henrique Tavares Aguiar

---

1. **A Biblioteca**

   O presente trabalho consiste na continuação de uma biblioteca para manipular grafos não dirigidos, mas agora com pesos nas arestas. E também com a implementação de um algoritmo para encontrar o menor caminho e distância entre dois vértices. E como na versão anterior, foi criada na linguagem de programação Python e testada em ambiente virtual linux (WSL2) Ubuntu 20.04.

   A biblioteca continua composta por uma classe concreta "Edge" e uma classe fachada "Graph".

   - **Edge**

     A classe Edge agora possui um atributo *weight* para representar o peso da aresta.

   - **Graph**

     Na classe Graph foi criada duas novas funções: *find_minimum_path* e *_dijkstra* que chamam as funções próprias das instâncias

     Para cada uma das representações foi criada uma classe própria que implementa os métodos acima.

     - Lista de Adjacência:

       Na implementação por lista de adjacência foi adicionado o peso e as funções necessárias para encontrar o menor caminho e distância entre dois vértices.

     - Matriz de Adjacência:

       A implementção da matriz foi alterada para uma matriz esparsa, mais precisamente, uma *dok_matrix* da biblioteca *scipy*, o qual implementa uma matriz esparsa através de um dicionário, o qual as chaves são as coordenadas. Tal escolha foi tomada, pois na implementação anterior, agora com a necessidade de armazenar os pesos das arestas, a matriz original consumiria uma quantidade exorbitante de memória. Ademais, assim como na lista, foram também criadas as funções necessárias para encontrar o menor caminho e distância entre dois vértices.

2. **Encontrar Menor Caminho entre Dois Vértices**

   - Grafo sem Peso:

Em grafos sem pesos, para encontrar o menor caminho é utilizada a BFS.

- Grafo com Peso:

  Para o caso do grafo com pesos, o algoritmo de Dijkstra foi utilizado para encontrar o menor caminho, as implementações variam um pouco entre os dois tipos de grafo, porém eles apenas diferem na forma em que os vérices são iterados. Além disso há uma verificação que checa se todos os pesos de um grafo são positivos para que o dijkstra seja executado corretamente, pois caso haja um ou mais valores negativos, o dijkstra pode não encontrar uma solução ótima, ou até mesmo ficar preso num loop infinito; caso hajam valores negativos um aviso é impresso no terminal do usuário.

Obs. Caso o vértice de origem ou de destino não exista no grafo um aviso é impresso no terminal do usuário.

3. **Estudo de Caso**

Para cada grafo do enunciado foi executada a função *find_minimum_path* a partir do vértice 1 até os vértices: 10, 100, 1000 e 10000. Além dos caminhos, para comparação da eficiência das implementações também foi calculado o tempo necessário para cada teste. Na tabela abaixo são apresentadas as médias dos tempos em segundos de execução dos testes para cada grafo.

Tipo	Grafo_1[1]	Grafo_2[2]	Grafo_3	Grafo_4	Grafo_5
Lista	$4.23 * 10^{-4} s$	$4.03 * 10^{-1} s$	$1.65 * 10^{-1} s$	$0.804 * 10^{-1} s$	$1.11s$
Matrix	$5.47 * 10^{-3} s$	$8.13 * 10^{-1} s$	$1.67 * 10^{1} s$	$5.83 * 10^{2} s$	$2.23 * 10^{3} s$

1. Os vértices 1000 e 10000 não se encontram no grafo
2. O vértice 10000 não se encontra no grafo