Algoritmos e Estrutura de Dados - IF969 2019.2

LISTA DE MONITORIA IV

Existem duas maneiras para implementar uma classe de Grafos, você pode representar as arestas através de *MATRIZ DE ADJACÊNCIA* ou através de *LISTA DE ADJACÊNCIA*. Implemente duas classes de Grafo, uma usando matriz e uma usando listas de adjacências. Ambas as classes devem funcionar tanto para grafos *DIRECIONADOS*, quanto para grafos *NÃO DIRECIONADOS*, assim como para *ARESTA COM PESO* e SEM PESO.

- Considere que seus vértices são números inteiros

- Seu método __init__ deve ser capaz de construir os grafos a partir de um objeto iterável que contenha tuplas de dois elementos (representando arestas de um grafo sem peso) ou três elementos (representando arestas de um grafo com peso). Por exemplo:

    - *g1 = ((0,1),(0,2),(0,3),(1,2),(2,3)),* onde cada um dos pares representa uma aresta entre os vértices nela contidos
    - ESPECIAL PARA GRAFO MATRIZ(Não obrigatório) : Utilize *KWARG* para conseguir construir um Grafo a partir de uma Matriz já pronta.

- Seus grafos deve conter os seguintes métodos construtores de python
    - __str__: Print do Grafo; Possivel exemplo

```
--- Sem peso ---
    0  1  2  3  4  5
0  [0, 0, 1, 0, 1, 1]
1  [0, 0, 0, 1, 0, 0]
2  [1, 0, 0, 1, 0, 0]
3  [0, 1, 1, 0, 0, 1]
4  [1, 0, 0, 0, 0, 0]
5  [1, 0, 0, 1, 0, 0]

--- Com peso ---
    0  1  2  3  4  5
0  [0, 0, 5, 0, 7, 3]
1  [0, 0, 0, 1, 0, 0]
2  [5, 0, 0, 6, 0, 0]
3  [0, 1, 6, 0, 0, 9]
4  [7, 0, 0, 0, 0, 0]
5  [3, 0, 0, 9, 0, 0]
```

```
--- Sem peso ---
Lista de adjacencia
0: [2, 4, 5]
1: [3]
2: [0, 3]
3: [1, 2, 5]
4: [0]
5: [0, 3]

--- Com peso ---
Lista de adjacencia
0: [(2, 5), (4, 7), (5, 3)]
1: [(3, 1)]
2: [(0, 5), (3, 6)]
3: [(1, 1), (2, 6), (5, 9)]
4: [(0, 7)]
5: [(0, 3), (3, 9)]
```

    - __repr__: Representação da sua Classe Grafo

- ○ __getitem__: Acessar um vetor por meio de indexação. Retorna, em forma de tupla, todas as arestas que se
  conecta ao vértice passado como parâmetro, se existir peso, retorna com o peso

● Seu Grafo deve conter um método que seja capaz de alterar o Grafo de *LISTA ADJACENTE* para *MATRIZ ADJACENTE* e o contrário também, de *MATRIZ* para *LISTA ADJACENTE*.

● Seu Grafo deve conter os seguintes métodos

- ○ **Adicionar um vértice:** Método para adicionar um novo vértice, ligando ele a vértice
- ○ **Adicionar uma aresta:** Método para ligar dos vértices do grafo entre si.
- ○ **Remover um vértice:** Remover um vértice específico passado como parâmetro. Se precisar do mesmo outra vez, utilize o método adicionar aresta. O vértice quando se é removido apenas tem todas a suas arestas "removidas"
- ○ **Remover uma aresta:** Método que remove uma aresta,específica, entre dois vértices
- ○ **Se dois vértices são ligados:** Método que retorna True se os vértices passado como parâmetro estiverem ligados entre si, e False para o contrário.
- ○ **Grau de entrada:** Método que retorne o grau de entrada de um vértice passado como parâmetro
- ○ **Grau de saída**: Método que retorna o grau de saída de um vértice passado como parâmetro
- ○ **Adjacente:** Método que retorna uma lista de todos os adjacentes de um vértice passado como parâmetro
- ○ **Menor Aresta:** Método que retorna a menor ou as menores arestas do Grafo e os vértices ligados a ela.
- ○ **Maior Aresta:** Método que retorna a maior ou as maiores arestas do Grafo e os vértices ligados a ela.

● **Implemente dois métodos** - Busca em largura e Busca em profundidade - esses métodos podem receber como parâmetro um grafo OU já está inserido em sua classe Grafo. Seus métodos deve retornar o vetor de antecessores a partir da busca feita. Suas funções devem funcionar tanto para Grafo de Lista quanto para Grafo de Matriz.

Minha dica pessoal a vocês. Utilizem esse site para auxiliar vocês. Me ajudou quando paguei essa cadeira, creio que vai ajudar a todos.