SIN110 – Algoritmos e Grafos
Prof. Rafael Frinhani

Atividade 3 - Tipos e Representação

Nome: Bruna Delmouro da Silva - 2021001809
Curso: Sistemas de Informação

---

1. Implementação

Link para o protótipo implementado:
https://github.com/brunadelmourosilva/UNIFEI-SIN110-Algorithms-Graphs/tree/master/class_3/grafo

Função 1 - tipo do grafo

Para esta função, foi implementado o seguinte código:

```
'''Tipo do grafo: Retorna o tipo do grafo representado por uma dada matriz de adjacências.
Entrada: matriz de adjacências
Saída: Integer (0 - simples; 1 - digrafo; 2 - multigrafo; 3 - pseudografo)
'''
def tipoGrafo(matriz):
    diagonalEhZerada = True
    numerosLimitadosAZeroOuUm = True
    matrizAssimetrica = True

    qtdVertices = np.shape(matriz)[0]
    for vi in range(0, qtdVertices):  # Para cada vértice vi
        for vj in range(vi + 1, qtdVertices):  # Para cada vértice vj
            if vi == vj:
                if matriz[vi][vj] != 0: #caso a diagonal possua valores diferentes de 0
                    diagonalEhZerada = False

            if matriz[vi][vj] > 1: #caso a matriz possua valores maiores que 1
                numerosLimitadosAZeroOuUm = False

            if matriz[vi][vj] == matriz[vj][vi]: #caso onde a posição dada e sua inversa sejam iguais
                matrizAssimetrica = False

    if diagonalEhZerada and numerosLimitadosAZeroOuUm: #grafo do tipo simples
        return 0
    elif matrizAssimetrica: #grafo do tipo digrafo
        return 1
    elif diagonalEhZerada and not numerosLimitadosAZeroOuUm: #grafo do tipo multigrafo
        return 2
    elif not diagonalEhZerada and not numerosLimitadosAZeroOuUm: #grafo do tipo pseudrografo
        return 3
```

Para o exemplo a seguir, escolhi o *dataset* ‘exemplo’ para a demonstração. Como o grafo é do tipo ‘simples’, o retorno será conforme segue a imagem do console:

```
NOME DA INSTÂNCIA: exemplo

[[0 1 1 0 0 1 0]
 [1 0 0 0 0 0 0]
 [1 0 0 1 1 1 0]
 [0 0 1 0 1 0 1]
 [0 0 1 1 0 0 0]
 [1 0 1 0 0 0 1]
 [0 0 0 1 0 1 0]]

IGRAPH U--- 7 9 --
+ attr: label (v)
+ edges:
0 -- 1 2 5     2 -- 0 3 4 5   4 -- 2 3       6 -- 3 5
1 -- 0         3 -- 2 4 6     5 -- 0 2 6

Tipo do grafo: 0
```

Função 3 - calcula densidade do grafo

Para esta função, foi implementado o seguinte código:

```python
'''Calcula densidade: Retorna o valor da densidade do grafo.
Entrada: matriz de adjacências
Saida: Float (valor da densidade com precisão de três casas decimais)
'''
def calcDensidade(self, matriz):
    # recebe o tipo do grafo para que a escolha do cálculo seja de acordo com o tipo retornado
    tipoGrafo = self.tipoGrafo(matriz)
    qtdVertices = np.shape(matriz)[0]

    if tipoGrafo == 0: #caso seja um grafo do tipo simples
        qtdArestas = 0
        for vi in range(0, qtdVertices):
            for vj in range(vi + 1, qtdVertices):
                if matriz[vi][vj] == 1: #se a posição for 1, significa que possui aresta
                    qtdArestas += 1 #realiza o calculo da quantidade de arestas

        return (2 * qtdArestas) / (qtdVertices * (qtdVertices - 1))

    if tipoGrafo == 1: #caso seja um grafo do tipo digrafo
        qtdArestas = 0
        for vi in range(0, qtdVertices):
            for vj in range(vi + 1, qtdVertices):
                if matriz[vi][vj] == 1:
                    qtdArestas += 1

        return qtdArestas / (qtdVertices * (qtdVertices - 1))
```

Como o grafo é do tipo ‘simples’, ou seja, o retorno da função tipoGrafo é 0, o cálculo será conforme segue a imagem do console:

```
Densidade do grafo: 0.42857142857142855
```

Função 4 - insere aresta

Para esta função, foi implementado o seguinte código:

```
'''Insere aresta: Insere uma aresta no grafo considerando o par de vértices vi e vj.
Entrada: matriz de adjacências, vi e vj (ambos são números inteiros que indicam o id do vértice)
Saída: matriz de adjacências (tipo numpy.ndarray) com a aresta inserida.
'''
def insereAresta(self, matriz, vi, vj):
    tipoGrafo = self.tipoGrafo(matriz)

    #se o grafo é do tipo digrafo(direcionado), adicionamos o valor 1 somente na posição indicada
    #se o grafo indicado for simples, multigrafo ou pseudografo, manteremos a simetria da matriz(caso válido somente
    #para grafos não direcionados)
    if tipoGrafo == 1:
        matriz[vi][vj] = 1
    else:
        matriz[vi][vj] = 1
        matriz[vj][vi] = 1

    return matriz
```

Para tal função, foi inserido o seguinte exemplo na classe main:

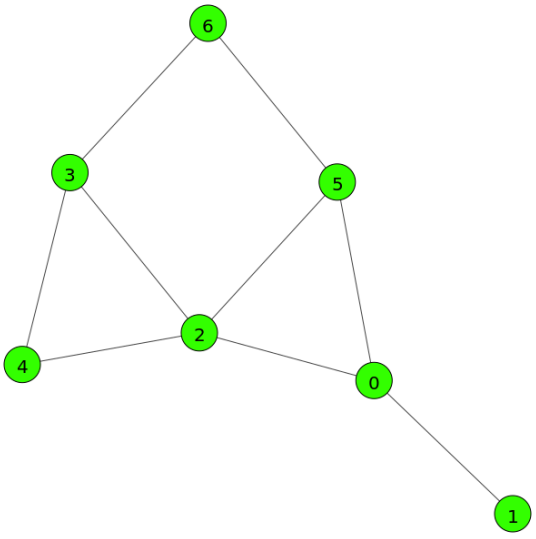
```
### FUNÇÃO 4 ###
linha = 1
coluna = 2
funcao4 = car.insereAresta(car, matriz, linha, coluna)
print('Insere aresta na posição {}x{}: \n {} \n'.format(linha, coluna, funcao4))

#criando um novo grafo para a viasualização da nova aresta
G = g.criaGrafo(funcao4)
vis.visualizarGrafo(True, G, instancia + " - insere aresta")

resultado = [instancia, funcao4]
ds.salvaResultado(resultado, instancia)
```

Como pode-se ver, realizei a chamada das funções que criam e salvam a imagem de um novo grafo, a fim de que possa-se obter uma comparação entre a imagem inicial do grafo ‘exemplo’ com a imagem após a inserção da nova aresta na posição 1x2.

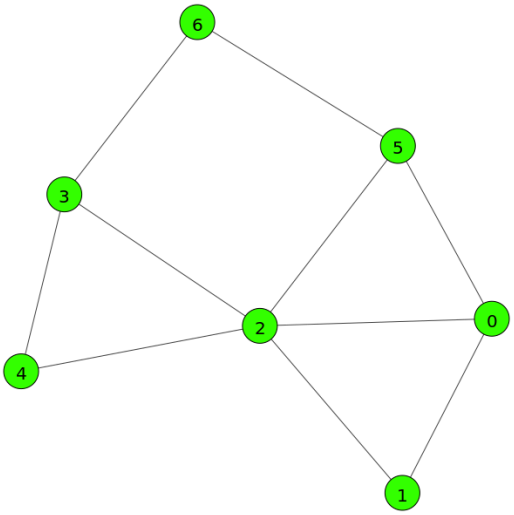
Antes da inserção da aresta:



```
NOME DA INSTÂNCIA: exemplo

[[0 1 1 0 0 1 0]
 [1 0 0 0 0 0 0]
 [1 0 0 1 1 1 0]
 [0 0 1 0 1 0 1]
 [0 0 1 1 0 0 0]
 [1 0 1 0 0 0 1]
 [0 0 0 1 0 1 0]]
```

Depois da inserção da aresta:




```
Insere aresta na posição 1x2:
 [[0 1 1 0 0 1 0]
 [1 0 1 0 0 0 0]
 [1 1 0 1 1 1 0]
 [0 0 1 0 1 0 1]
 [0 0 1 1 0 0 0]
 [1 0 1 0 0 0 1]
 [0 0 0 1 0 1 0]]
```

<u>Função 5 - insere vértice</u>

Para esta função, foi implementado o seguinte código:

```python
'''Insere vértice: Insere um vértice no grafo.
Entrada: matriz de adjacências, vi (número inteiro que indica o id do vértice)
Saída: matriz de adjacências (tipo numpy.ndarray) com o vértice inserido.
'''
def insereVertice(matriz, vi):
    shape = matriz.shape #recebe o resultado do número de linhas e colunas da atual matriz
    novaMatriz = numpy.zeros((shape[0] + 1, shape[1] + 1)) #cria nova matriz(números zeros) com mais uma linha e coluna
    qtdVertices = np.shape(matriz)[0] #recebe a quantidade de vértices

    #a nova matriz recebe os valores da matriz antiga de acordo com as posições da iteração
    for vi in range(0, qtdVertices):
        for vj in range(0, qtdVertices):
            novaMatriz[vi][vj] = matriz[vi][vj]

    return novaMatriz
```

Tal estratégia foi implementada para que, uma nova linha e uma nova coluna sejam inseridas(com valores 0 em cada posição) na nova matriz criada para que, posteriormente, possa ser gerada uma imagem com o novo vértice.

Para tal função, foi inserido o seguinte exemplo na classe main:

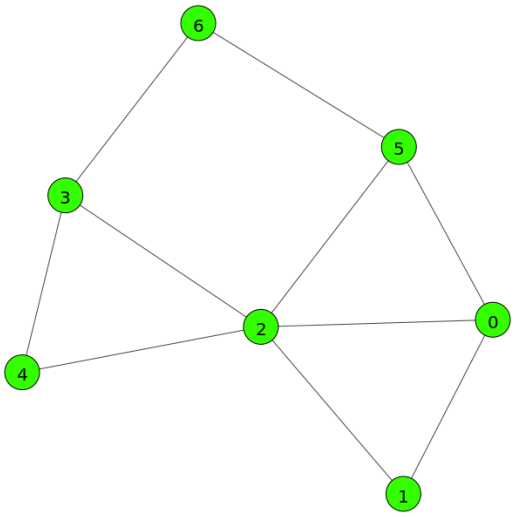
```
### FUNÇÃO 5 ###
funcao5 = car.insereVertice(matriz, 1)
print('Insere vértice: \n {} \n'.format(funcao5))

# criando um novo grafo para a viasualização da nova aresta
G = g.criaGrafo(funcao5)
vis.visualizarGrafo(True, G, instancia + " - insere vértice")

resultado = [instancia, funcao5]
ds.salvaResultado(resultado, instancia)
```

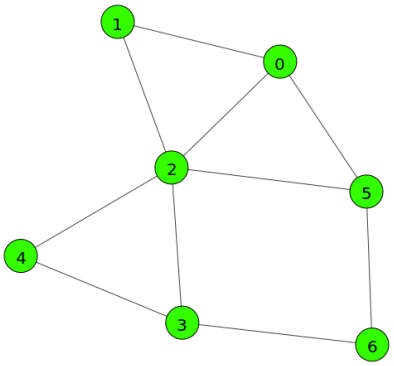
Como pode-se ver, realizei a chamada das funções que criam e salvam a imagem de um novo grafo, a fim de que possa-se obter uma comparação entre a imagem gerada anteriormente, com a chamada da função 'insere aresta' e a imagem após a inserção do novo vértice.

Antes da inserção do vértice:




```
Insere aresta na posição 1x2:
 [[0 1 1 0 0 1 0]
 [1 0 1 0 0 0 0]
 [1 1 0 1 1 1 0]
 [0 0 1 0 1 0 1]
 [0 0 1 1 0 0 0]
 [1 0 1 0 0 0 1]
 [0 0 0 1 0 1 0]]
```

Depois da inserção do vértice:




```
Insere vértice:
 [[0. 1. 1. 0. 0. 1. 0. 0.]
 [1. 0. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 1. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

Função 6 - remove aresta

Para esta função, foi implementado o seguinte código:

```python
'''Remove aresta: Remove uma aresta do grafo considerando o par de vértices vi e vj.
Entrada: matriz de adjacências, vi e vj (ambos são números inteiros que indicam os ids dos vértices)
Saída: matriz de adjacências (tipo numpy.ndarray) com a aresta removida.
'''
def removeAresta(self, matriz, vi, vj):
    tipoGrafo = self.tipoGrafo(matriz)

    # se o grafo é do tipo dígrafo(direcionado), adicionamos o valor 0 somente na posição indicada
    # se o grafo indicado for simples, multigrafo ou pseudografo, manteremos a simetria da matriz(caso válido somente
    # para grafos não direcionados)
    if tipoGrafo == 1:
        matriz[vi][vj] = 0
    else:
        matriz[vi][vj] = 0
        matriz[vj][vi] = 0

    return matriz
```

Para tal função, foi inserido o seguinte exemplo na classe main:

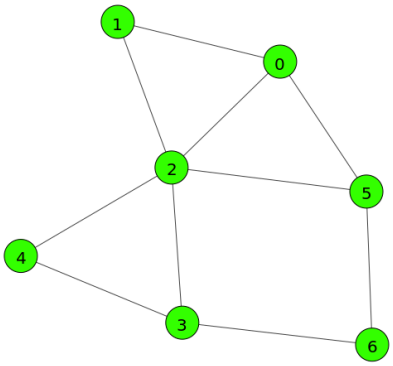
```python
### FUNÇÃO 6 ###
linha = 1
coluna = 2
funcao6 = car.removeAresta(car, funcao5, linha, coluna)
print('Remove aresta: \n {} \n'.format(funcao6))

G = g.criaGrafo(funcao6)
vis.visualizarGrafo(True, G, instancia + " - remove aresta")

resultado = [instancia, funcao6]
ds.salvaResultado(resultado, instancia)
```

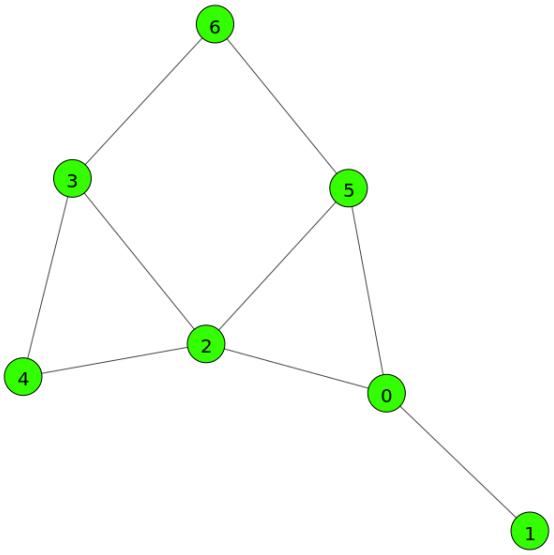
Como pode-se ver, realizei a chamada das funções que criam e salvam a imagem de um novo grafo, a fim de que possa-se obter uma comparação entre a imagem gerada anteriormente, com a chamada da função 'insere vértice' e a imagem após a remoção da aresta na posição 1x2.

Antes da remoção da aresta:




```
Insere vértice:
 [[0. 1. 1. 0. 0. 1. 0. 0.]
 [1. 0. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 1. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

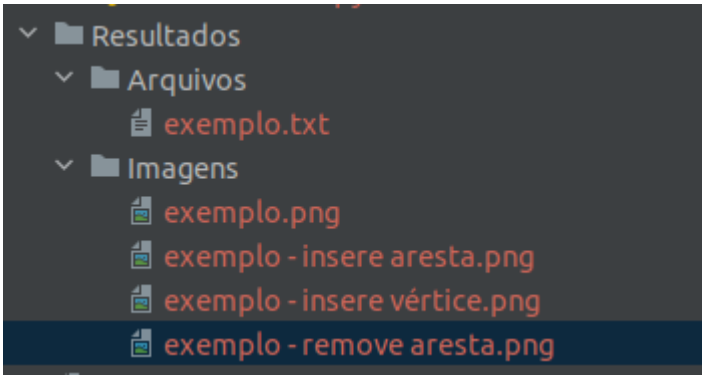
Depois da remoção da aresta:



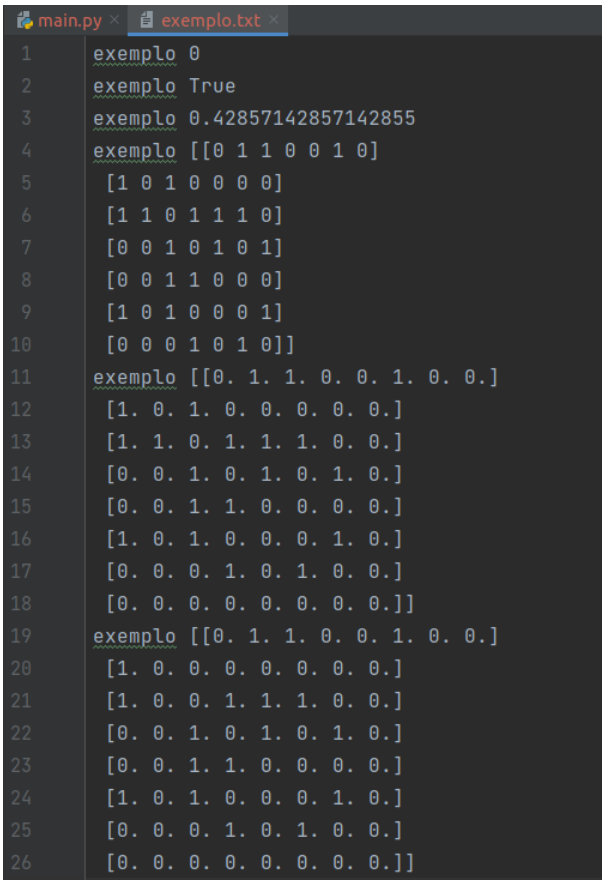
```
Remove aresta:
 [[0. 1. 1. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

2. Estrutura dos arquivos

Vale ressaltar que, assim que o código é executado na classe main, serão gerados os arquivos de imagem referente ao grafo inicial e posteriormente, às imagens referentes à inserção da aresta, inserção do vértice e remoção da aresta. Ademais, será gerado o arquivo em formato .txt, de acordo com o *dataset* passado na classe main.



Por fim, segue a imagem do arquivo .txt final gerado pelo *dataset* 'exemplo':



3. Dificuldades

Obtive certas dificuldades em relação à lógica, para realizar a implementação da função ‘remove vértice’ já que, para realizar a remoção de um determinado vértice, após a essa ação, seria necessário trazer tanto as linhas quantos as colunas para os lugares em que os mesmos ficassem vazios após a remoção. Realizei algumas pesquisas para sanar as dúvidas dos problemas gerados, mas não obtive muito progresso.