Atividade – Grafos – Análise e Projeto de Sistemas

Marcos Vinicius Januário da Silva

Mestrando de Engenharia de Computação – Universidade Estadual do Maranhão (UEMA)

mvjanuario@hotmail.com

1. Entrada do Arquivo

O arquivo é dado com o nome de "entrada.txt", contendo as informações para execução. É necessário que o arquivo esteja com este nome, pois o código está programado para receber dessa forma. Abaixo segue código desta etapa.

```
14    #leitura Grafo do txt
15    #O arquivo precisa estar na mesma pasta do .exe
16    grafo = open('entrada.txt', 'r')
```

Para teste foi utilizado um arquivo entrada.txt próprio criado para verificar as funções. Abaixo segue o .txt usado para as verificações.

```
D
v1,v2
v1,v3
v2,v3
v2,v4
v4,v2
v2,v2
v1,v4
v5,v3
```

2. Armazenamento na Estrutura de Dados

A Estrutura de Dados escolhida para guardar os dados do Grafo foi a Matriz de Adjacência. Esta Estrutura de Dados é construída tomando os dados do arquivo ".txt", com os vértices. O total de linhas e colunas da matriz é a quantidade de vértices do Grafo, ou seja, a matriz é de "v por v", onde v é o número de vértices.

Abaixo segue o código que realiza a operação de armazenamento.

```
39  #criando Matriz Adjacencia a partir dos dados
40  quantidadeVertices = len(listaVertices)
41  matrizAdjacencia = numpy.zeros(shape=(quantidadeVertices,quantidadeVertices))
42
43  for v0 in listaVertices:
44      #matrizAdjacencia
45      print(v0)
46      for l in listaAdjacencia:
47          if v0 == l[0]:
48              print(l[0], l[1])
49              if tipo == "D":
50                  matrizAdjacencia[listaVertices.index(l[0])][listaVertices.index(l[1])] = 1
51              if tipo == "ND":
52                  matrizAdjacencia[listaVertices.index(l[0])][listaVertices.index(l[1])] = 1
53                  matrizAdjacencia[listaVertices.index(l[1])][listaVertices.index(l[0])] = 1
```

Primeiramente, verificasse qual o tipo de Grafo. Se for Dirigido, ou seja, tipo “D”, ao salvar a adjacência na matriz colocasse “1” apenas na linha v1 e coluna v2, sendo v1 o vértice de origem e v2 o vértice destino. Quando o tipo de Grafo é Não Dirigido, ou seja, “ND”, ao salvar a adjacência na matriz colocasse “1” tanto na linha v1 e coluna quanto na linha v2 e coluna v1, pois a aresta não possui direção.

3. Parte 1 – Tarefas

Nesta seção são tratadas as resoluções das tarefas propostas para a Parte 1 da Atividade. A forma de interação com o usuário se dá por meio de um **menu** básico de seleção a ser realizada.

3.1 Apresentar se dois vértices vX e vY são ou não adjacentes

Esta tarefa tem objetivo de receber dois vértices v1 e v2 e verificar pela Matriz Adjacência se estes dois vértices são adjacentes ou não, como a imagem abaixo demonstrando no código. Para fazer esta tarefa é necessário selecionar a opção “1” no menu.

```
78                  if l1 != l2:
79                      adjacente = 0;
80                      for m in range(len(matrizAdjacencia)):
81                          if m == l1:
82                              for n in range(len(matrizAdjacencia[m])):
83                                  if (n == l2) & ((matrizAdjacencia[m][n] == 1) | (matrizAdjacencia[n][m] == 1)):
84                                      adjacente = 1
85                      if adjacente == 1:
86                          print("\nSão adjacentes")
87                      else:
88                          print("\nNão são adjacentes")
89                  else:
90                      print("\nOs dois vértices são iguais")
```

A execução desta tarefa segue o fluxo de entrada dos dados, que não mostrado na imagem. Logo após, existe a verificação na linha 78 para verificar se os vértices passados não são iguais. Se forem iguais, a execução não é feita e mostra a mensagem “Os dois vértices são iguais”. Se forem diferentes, o fluxo continua.

Para verificar a adjacência, a Matriz Adjacência é percorrida por linhas m, na linha 80, e por colunas n, na linha 82. As linhas 83 a 84 realizam a tarefa, verificam se existe “1” na posição [m][n] e na posição [n][m] da Matriz, caracterizando assim que são adjacentes, pois existe ligação entre os mesmos. Se os valores encontrados forem “0” significa que não são adjacentes.

Existem verificações para o caso de o usuário digitar valores errados, porém a imagem acima somente mostra parte da função de verificar a adjacência.

Abaixo segue uma imagem mostrando um exemplo de execução:

```
Selecione uma opção:
1

Insira o vértice 1:
v1

Insira o vértice 2:
v3

São adjacentes
```

3.2 Calcular o grau de um vértice qualquer

Esta tarefa tem objetivo de receber um vértice v e verificar pela Matriz Adjacência qual o grau do vértice em questão, ou seja, quantos vértices estão ligados v, como a imagem abaixo demonstrando no código. Para fazer esta tarefa é necessário selecionar a opção “2” no menu.

```
105             grau = 0
106             for l in listaAdjacencia:
107                 if (l[0] == v) | (l[1] == v):
108                     grau = grau + 1
109             print("\nGrau: ", grau)
```

A função segue o fluxo recebendo o vértice v que se deseja conhcer o grau. Com isso, o laço “for” na linha 106 percorre a Lista de Adjacências dos vértices, e o “if” na linha 107 verifica se o vértice v possui ligações com outros vértices, analisando a Lista de Adjacência. Se existir ligação, a variável “grau” aumenta em “1”. Ao final é apresentado o grau do vértice. Se não existir nenhuma ligação, o valor apresentado é 0.

Existem verificações para o caso de o usuário digitar valores errados, porém a imagem acima somente mostra parte da função de verificar o grau do vértice.

Abaixo segue uma imagem mostrando um exemplo de execução:

```
Selecione uma opção:
2

Insira o vértice:
v3

Grau:  3
```

3.3 Buscar todos os vizinhos de vértice qualquer

Esta tarefa tem objetivo de receber um vértice v e verificar pela Matriz Adjacência quais os vizinhos do vértice em questão, ou seja, quais os vértices ligados v, como a imagem abaixo demonstrando no código. Para fazer esta tarefa é necessário selecionar a opção "3" no menu.

```
121             for m in range(len(matrizAdjacencia)):
122                 if m == l:
123                     for n in range(len(matrizAdjacencia[m])):
124                         if (m != n) & ((matrizAdjacencia[m][n] == 1) | (matrizAdjacencia[n][m] == 1)):
125                             print(listaVertices[n])
```

Nas linhas 121 e 122 o código procura pelo vértice v na Matriz Adjacência. A linha 123 percorre as adjacências do vértive v, e a linha 124 verifica se o vértice em questão está ligado a v. Os vértices ligados são mostrado na tela pela linha.

Existem verificações para o caso de o usuário digitar valores errados, porém a imagem acima somente mostra parte da função de buscar os vizinhos do vértice.

Abaixo segue uma imagem mostrando um exemplo de execução:

```
Selecione uma opção:
3

Insira o vértice:
v2
v1
v3
v4
```

3.4 Visitar todas as arestas do grafo

Esta tarefa tem o objetivo simples de mostrar as arestas do Grafo, passando por todos os vértices, como a imagem abaixo demonstrando no código. Para fazer esta tarefa é necessário selecionar a opção "4" no menu.

```
129     #Visitando arestas
130     elif opcao == '4':
131         print("\nArestas:\n")
132         for l in listaAdjacencia:
133             print("De ", l[0], "Para", l[1])
```

A execução funciona listando os itens da Lista de Adjacência nas linhas 132 e 133.

Abaixo segue uma imagem mostrando um exemplo de execução:

```
Selecione uma opção:
4

Arestas:

De  v1 Para v2
De  v1 Para v3
De  v2 Para v3
De  v2 Para v4
De  v4 Para v2
De  v2 Para v2
De  v1 Para v4
De  v5 Para v3
```

4. Parte 2 – Apresentação Gráfica

Esta etapa da Atividade possui o objetivo de mostrar graficamente o Grafo rebido. No caso do presente trabalho, como foi utilizada a mesma linguagem para as tarefas e para a parte gráfica, o ciclo de execução continua sem finalizar. Para mostrar o Grafo pela imagem, deve-se selecionar a opção “5” no menu.

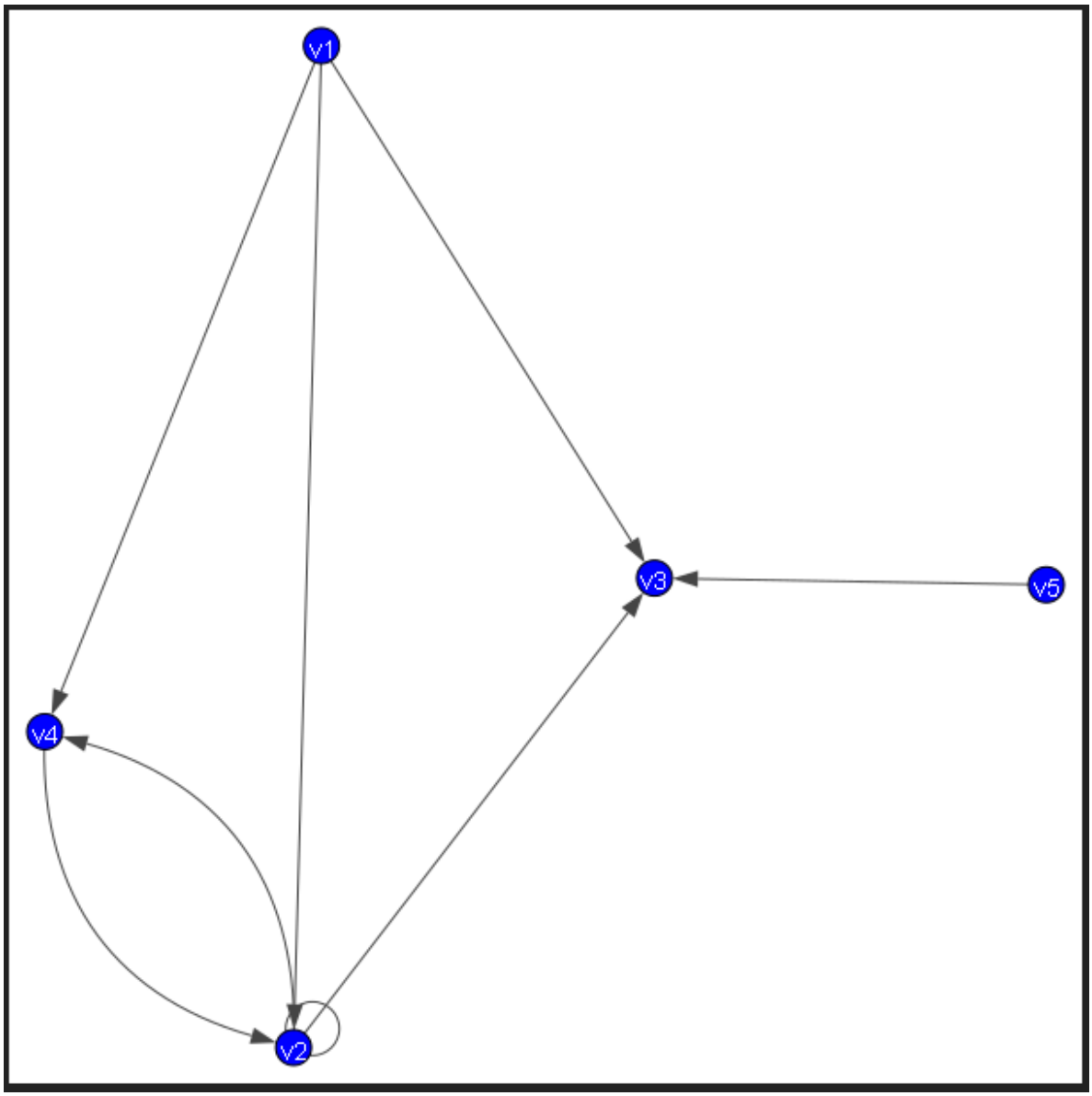
Abaixo está o código referente a esta etapa.

```
135     #Visualizar graficamente
136     elif opcao == '5':
137         print("\n\nAguarde. Preparando a visualização gráfica do Grafo...")
138         print("\nA imagem será aberta em aplicativo do computador...")
139         if tipo == "D":
140             g = Graph(directed=True)
141         else:
142             g = Graph()
143         g.add_vertices(len(listaVertices))
144         g.vs["name"] = listaVertices
145 
146         for l in listaAdjacencia:
147             g.add_edges([(listaVertices.index(l[0]),listaVertices.index(l[1]))])
148 
149         g.vs["label"] = g.vs["name"]
150         plot(g, vertex_color = "blue", vertex_label_color = "white")
```

Na função de exibição gráfica, primeiramente é verificado se o Grafo é ou não dirigido, nas linhas 139 até 142. Se for dirigido o Grafo é criado adequadamente e da mesma forma se não for dirigido.

Seguindo, os vértices são nomeados, de acordo com os nomes passados no arquivo entrada.txt, realizado nas linhas 143 e 144. Após isso, os vértices são adicionados cada um dentro do Grafo que está sendo construído, pelas linhas 146 e 147. A linhas 149 e 150 possuem a função de preparar e exibir graficamente o Grafo.

Abaixo segue uma imagem mostrando um exemplo de Grafo plotado:



5. Conclusão

Portanto, percebe-se que os Grafos podem ser muito explorados e estudados, sendo utilizados em diversas áreas. Usando a linguagem Python foram realizadas algumas tarefas e atividades através da manipulação de arquivo .txt representando Grafos.