

# MESTRADO EM ENGENHARIA DE COMPUTAÇÃO E SISTEMAS/PECS

Elton de Sousa e Silva

<sup>3</sup>Engenharia da Computação

Universidade Estadual do Maranhão (UEMA) – São Luís, MA – Brasil

eltonss.eng@gmail.com

## 1. Introdução

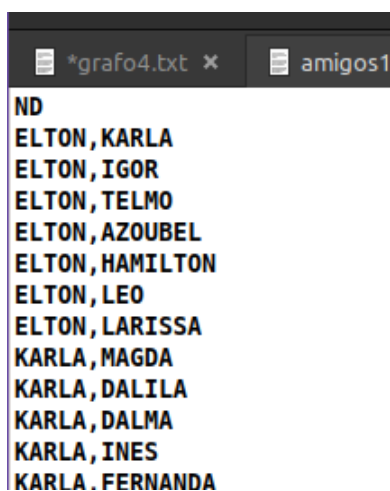
Teoria dos grafos é uma área da matemática que estuda as relações entre objetos de um determinado conjunto. Para isso é usada uma estrutura chamada de grafos  $G(V, E)$  onde  $V$  é um conjunto de vértices que por definição não pode ser vazio e  $E$  é um conjunto de arestas.

1. Este programa tem como objetivo facilitar a solução dos problemas abaixo:

- Receber dois vértices ( $V_x$  e  $V_y$ ) e apresentar se são ou não adjacentes.
- Calcular o grau de um vértice qualquer
- Buscar todos os vizinhos de um vértice qualquer
- Visitar todas as arestas do grafo
- Plotar a apresentação gráfica do grafo.

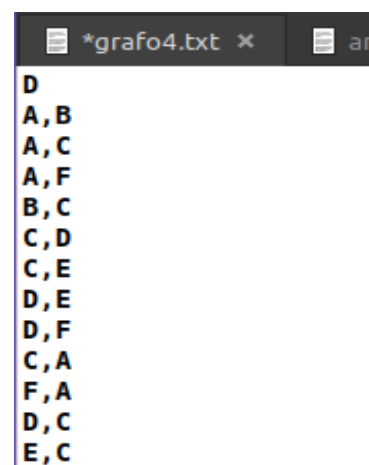
2. Entradas do Software:

1. Arquivo de texto no formato (Extensão .txt), exemplo:



```
*grafo4.txt x  amigos1
ND
ELTON, KARLA
ELTON, IGOR
ELTON, TELMO
ELTON, AZOUBEL
ELTON, HAMILTON
ELTON, LEO
ELTON, LARISSA
KARLA, MAGDA
KARLA, DALILA
KARLA, DALMA
KARLA, INES
KARLA, FERNANDA
```

Figura 1 Grafo não direcionado



```
*grafo4.txt x  an
D
A, B
A, C
A, F
B, C
C, D
C, E
D, E
D, F
D, F
C, A
F, A
D, C
E, C
```

Figura 2 Grafo direcionado

## 2. Linguagem e bibliotecas Utilizadas

Para desenvolvimento do software foi utilizado o Python e foi adicionado as seguintes bibliotecas padrão ao sistema.

### 2. Numpy

Suporte a *arrays* e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

### 3. Networkx

Adequado para operação em grandes gráficos foi utilizado no auxílio do desenho do grafo.

### 4. Matplotlib

Usado para criação de gráficos e visualização de dados

### 5. Pathlib

Acesso a arquivos externos.

## 3. Funções principais desenvolvidas no software

### 6. **getArestasFromFile()**

Dado um Arquivo (arquivo.txt) Retorna Arestas e o Tipo de Grafo

### 7. **visitarVertices(\_vertices, \_MatrizAdjacencia)**

Dado uma lista de vértices e a matriz de adjacência visita todos os vértices

### 8. **calcularGrauVertices(\_vertice, \_vertices, \_MatrizAdjacencia, \_tipoGrafo)**

Dado um vértice calcula o grau desse vértice.

### 9. **verAdjacenciaVertices(\_vertice1, \_vertice2 , \_vertices, \_MatrizAdjacencia)**

Dado duas vértices verifica se são adjacentes

### 10. **buscaVerticesVizinhas(\_vertice, \_vertices, \_MatrizAdjacencia, \_tipoGrafo)**

Dado um vértice busca seus vizinhos.

### 11. **desenhaDiGrafo(\_arestas, \_tipoGrafo)**

Dado uma lista de arestas e tipo de grafo desenha o gráfico do grafo.

#### 4. Software - análise de partes do código

- ❖ Trecho Código 1 – Leitura do Arquivo.txt e separação do tipo de grafo e lista de vértices
  - Linha 33 solicita o endereço do arquivo
  - Linha 35 verifica a existência do arquivo
  - Linha 45 abre o arquivo em formato .txt
  - Linha 47-48 carrega os dados do arquivo para uma lista
  - Linha 52 salva separadamente em uma lista o tipo de grafo e a lista de vértices

```

32 while(endOK):
33     endfile = input("Qual a localização do arquivo: ")
34     my_file = Path(endfile)
35     if (my_file.is_file()==False):
36         os.system('clear')
37         print('Erro ao Localizar o Arquivo')
38         print('Verifique o endereço digitado.')
39         endOK = True
40     else:
41         os.system('clear')
42         print('Arquivo Aberto: ', endfile)
43         print('')
44         endOK = False
45 _file = open(endfile,'r')
46 linhasarquivos = []
47 for linha in _file:
48     linhasarquivos.append(linha.split()[0]);
49 _file.close()
50 tipografo = linhasarquivos[0]
51 del linhasarquivos[0]
52 _arestas = [tipografo, linhasarquivos]
53 return _arestas

```

#Na Posicao 0 o tipo de Grafo na Posicao 1 as arestas

```
Bom dia Professor Blz..
Qual a localização do arquivo: amigos1.txt
```

### Tipo Grafo Não Direcionado

```
Lista de Arestas
['ELTON,KARLA','ELTON,IGOR','ELTON,TELMO','ELTON,AZOUBEL','ELTON,HAMILTON','ELTON,INES','KARLA,FERNANDA','IGOR,ENIO','IGOR,SILVIA','IGOR,PAULO','AZOUBEL,FERNANDA','HAMILTON,HENRIQUE','LEO,LARISSA','LEO,BATISTA','BATISTA,INES']
```

- ❖ Trecho de Código 2 – Recebe uma lista de vértices, arestas e tipo de grafo e gera uma matriz de adjacências
- Linha 62 usa a biblioteca numpy para criar uma matriz nxn onde n é a quantidade de vértices
- Linha 63 verifica o tipo de grafo (Direcionado ou não direcionado) e preenche a matriz de acordo com o tipo.
- Linha 76 retorna a matriz de adjacências

```

61 def criarMatrizAdjacencia(_vertices, _arestas, _tipoGrafo):
62     _matriz = np.zeros(shape=(len(_vertices), len(_vertices)))
63     if _tipoGrafo == 'ND':
64         for j in _arestas:
65             _itemExplodido = j.split(',')
66             linha = _vertices.index(_itemExplodido[0])
67             coluna = _vertices.index(_itemExplodido[1])
68             _matriz[linha][coluna] = 1
69             _matriz[coluna][linha] = 1
63         else:
64             for j in _arestas:
65                 _itemExplodido = j.split(',')
66                 linha = _vertices.index(_itemExplodido[0])
67                 coluna = _vertices.index(_itemExplodido[1])
68                 _matriz[linha][coluna] = 1
69                 _matriz[coluna][linha] = 1
70     return _matriz

```

**Tipo Grafo Não Direcionado**

Lista de Arestas  
['ELTON,KARLA', 'ELTON,IGOR', 'ELTON,TELMO', 'ELTON,AZOUBEL', 'ELTON,HAMILTON', 'ELTON,KARLA,FERNANDA', 'IGOR,ENIO', 'IGOR,SILVIA', 'IGOR,PAULO', 'AZOUBEL,FERNANDA', 'ENIO,ALEXANDRE', 'HAMILTON,HENRIQUE', 'LEO,LARISSA', 'LEO,BATISTA', 'BATISTA,INES']

[illegible]

❖ Trecho de Código 3 – Visita todos os vértices.

- Linha 79 usa a biblioteca numpy para criar uma matriz nxn onde n é a quantidade de vértices nessa matriz será armazenados os vértices que já foram visitados.
- Linha 86 verifica se o vértice é adjacente e já foi visitado.
- Linha 87 caso não visitado salva como visitado e imprimir o vértice conforme figura abaixo.

```

78 def visitarVertices(_vertices, _MatrizAdjacencia):
79     isVisitado = np.zeros(shape=(len(_vertices),len(_vertices)))
80     x = 0
81     for i in _vertices:
82         y = 0
83         for j in _vertices:
84             if y == 0:
85                 print ('Vertice ' + i)
86                 if _MatrizAdjacencia[x][y] == 1 and isVisitado[y][x] !=1:
87                     isVisitado[x][y] =1
88                     print ('          ' + j + ' Visitado')
89                 y = y+1
90             x = x+1
91     return

```

```

Opção: 4
Vertice ALEXANDRE
Vertice HAMILTON Visitado
Vertice AZOUBEL
Vertice ELTON Visitado
Vertice FERNADA Visitado
Vertice JOAO Visitado
Vertice MONICA Visitado
Vertice PEDRO Visitado
Vertice SERQUEIRA Visitado
Vertice BATISTA
Vertice INES Visitado
Vertice LEO Visitado
Vertice DALILA
Vertice KARLA Visitado
Vertice DALMA
Vertice KARLA Visitado
Vertice ELTON

```

❖ Trecho de Código 4 – Calcula grau do Vértice.

- Linha 114 e 124 verifica o tipo de Grafo (Direcionado ou Não direcionado)
- Linha 115 e 125 verifica se existe a vértice digitada
- Linha 121 Soma os números de 1 na planilha de adjacências calculando o grau do vértice em grafos não direcionado.
- Linha 131 e 132 Soma os números de 1 na planilha de adjacências calculando o grau do vértice (grau saída e entrada) em grafos direcionado.

```

113 def calcularGrauVertices(_vertice, _vertices, _MatrizAdjacencia, _tipoGrafo): #Calcula o Grau de um Vertice
114     if _tipoGrafo == 'ND':
115         if _vertice not in _vertices:
116             return 'A Vertice ' + _vertice + ' não pertence ao grafo'
117         linha = _vertices.index(_vertice)
118         print(linha)
119         linhaGrau = _MatrizAdjacencia[linha]
120         colunaGrau = _MatrizAdjacencia.T[linha]
121         _grau = np.sum(linhaGrau)
122
123         return 'A vertice ' + str(_vertice) + ' tem grau ' + str(_grau)
124     else:
125         if _vertice not in _vertices:
126             return 'A Vertice ' + _vertice + ' não pertence ao grafo'
127         linha = _vertices.index(_vertice)
128         print(linha)
129         linhaGrau = _MatrizAdjacencia[linha]
130         colunaGrau = _MatrizAdjacencia.T[linha]
131         _grausaida = np.sum(linhaGrau)
132         _grauentrada = np.sum(colunaGrau)
133         return 'A vertice ' + str(_vertice) + ' tem: ' + str(_grausaida) + ' de Saída e ' + str(_grauentrada) + ' de Entrada'

```

```

Entre com a vertice: ELTON
5

A vertice ELTON tem grau 7.0

```

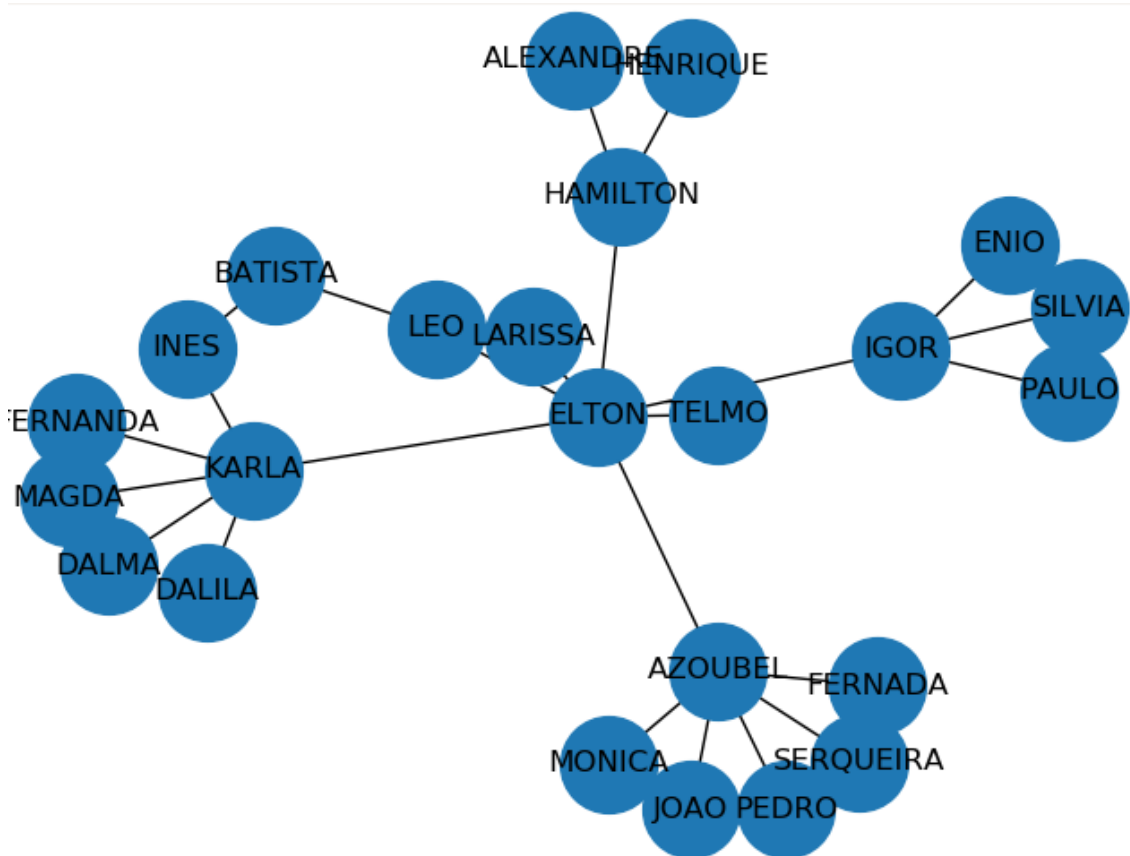
❖ Trecho de Código 5 – Desenha o Grafo.

- Linha 172 Cria um objeto G que recebe o grafo
- Linha 177 Função que converte a lista de vértice no formato aceito pela API.
- Linha 178 Adiciona os vértices no objeto G
- Linha 179 Formata o objeto (Vértices) para plotar.
- Linha 180 Desenhar o grafo
- Linha 181 Mostrar o grafo

```

171 def desenhaDiGrafo(_arestas, _tipoGrafo):
172     G = nx.DiGraph()
173     if _tipoGrafo == 'ND':
174         print('Entrou nao direct')
175         G = nx.Graph()
176
177     newVertices = parseVertices(_arestas)
178     G.add_edges_from(newVertices)
179     nx.draw(G, with_labels=True, node_size = 1500)
180     plt.draw()
181     plt.show()
182     return

```



## 7. Código fonte e execução do Software

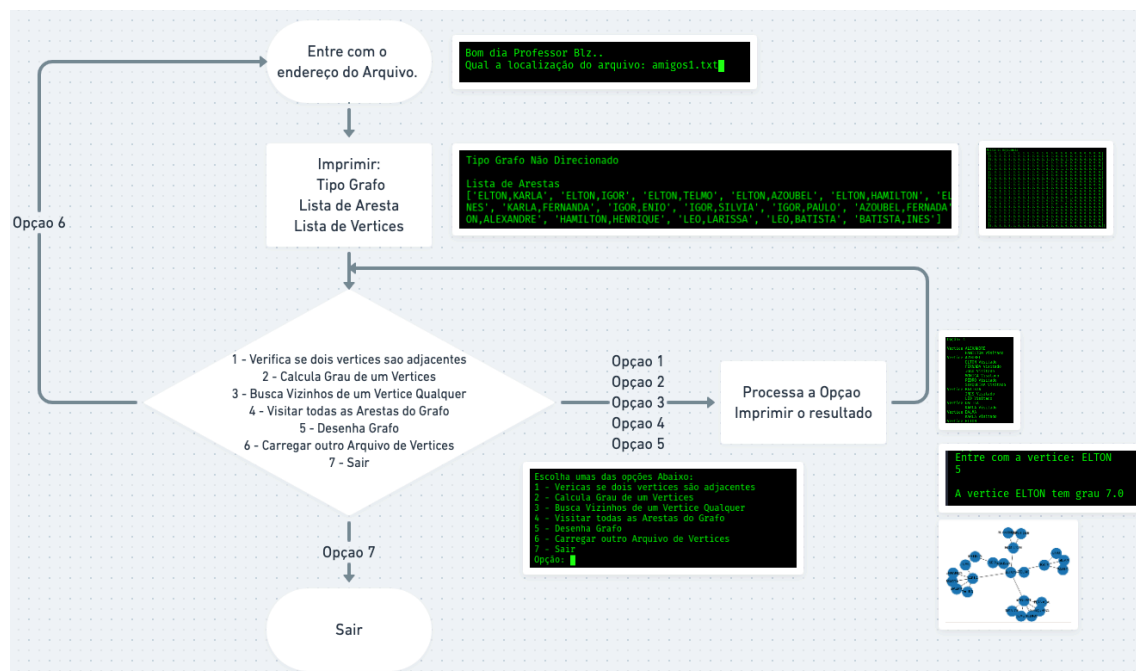
O código fonte é composto pelos seguintes arquivos.

- main.py – Arquivo principal para ser executado quando pelo interpretador.
- funcGrafo.py – Arquivo contendo todas as funções principais
- InitGrafo.py – Arquivo contendo um script de inicialização do arquivo
- dist/main.exe – Arquivo .exe para ser executado diretamente no Windows

O software pode ser executado de duas formas.

- Através de um interpretador de código python
  - Para iniciar o software deve ser executado o arquivo main.py
- Através do código executável exe (Windows).
  - Main.exe

Independente da forma de execução o aplicativo tem o seguinte fluxograma.



## **5. Referencia**

P. Feofiloff, Y. Kohayakawa, Y. Wakabayashi, Uma Introdução Sucinta à Teoria dos Grafos, 2004 .

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Algoritmos Teoria e Prática. 2. ed. São Paulo Editora Campus, 2002 .