

Para o desenvolvimento do nosso trabalho, utilizamos a linguagem `python`, que tem um conjunto de bibliotecas que nos ajudaram bastante na resolução dos problemas propostos.

O trabalho foi desenvolvido com orientação a objetos, por isso, teremos três classes a serem analisadas: `main`, `grafo` e `menu`.

Neste relatório iremos descrever o que cada uma das classes listadas possuem e qual é a função de cada uma¹.

Main

1. Inicialmente foi criada uma `main` que basicamente chama as outras classes que farão todo o resto.

```
1  #!/ python3
2  # coding: utf-8
3
4
5  from Classes.grafo import Grafo
6  from Classes.menu import Menu
7
8  grafo = Grafo()
9  m = Menu()
10 m.mostra_menu(grafo)
```

Menu

1. A classe `menu` é uma classe que o próprio nome já diz, mostra o menu para o usuário.

```
1  import subprocess as sp
2  from PIL import Image
3
4
5  class Menu:
6     Classe menu
7
8     def __init__(self):
9         self.vertice = 0
10        self.opcao = 0
```

Primeiramente importamos as libs `subprocess` e `PIL` que tem por finalidade (nesse código) limpar a tela a cada ENTER que o usuário der e mostrar a imagem gerada pelo método `grafo_visual()` da classe `Grafo` (será explicada mais adiante) respectivamente.

Depois criamos a classe `Menu` e chamamos o construtor `def __init__(self)` que serve para iniciarmos os atributos globais da nossa classe.

¹Algumas funções foram ocultadas para evitar que o relatório ficasse massivo.

2. Agora criamos o menu onde mostrará as opções para o usuário escolher:

```
1  def mostra_menu(self, grafo):
2      Mostrar o menu com todas as opcoes na tela
3
4      while self.opcao != -1:
5          grafo.mostra_dados(grafo.matriz)
6          print('\t', end='')
7          print('='*34)
8          print('\t=', f'{Menu:^30}', '=')
9          print('\t', end='')
10         print('='*34)
11         print(f'\t| {01 - Inserir vertice:<30}', '|')
12         print(f'\t| {02 - Inserir aresta:<30}', '|')
13         print(f'\t| {03 - Seu grafo é completo?:<30}', '|')
14         print(f'\t| {04 - Gerar grafo complementar:<30}', '|')
15         print(f'\t| {05 - Número de componentes:<30}', '|')
16         print(f'\t| {06 - Seu grafo é uma árvore?:<30}', '|')
17         print(f'\t| {07 - Busca em largura:<30}', '|')
18         print(f'\t| {08 - Busca em profundidade:<30}', '|')
19         print(f'\t| {09 - Algoritmo de Dijkstra:<30}', '|')
20         print(f'\t| {10 - Gerar grafo aleatório:<30}', '|')
21         print(f'\t| {11 - Visualizar grafo:<30}', '|')
22         print(f'\t| {-1 - Sair:<30}', '|')
23         print('\t', end='')
24         print('='*34)
```

Feito isso um menu como este será mostrado na tela:

```
=====
=                               =
=                               Menu                               =
=====
| 01 - Inserir vertice          |
| 02 - Inserir aresta          |
| 03 - Seu grafo é completo?   |
| 04 - Gerar grafo complementar |
| 05 - Número de componentes   |
| 06 - Seu grafo é uma árvore? |
| 07 - Busca em largura        |
| 08 - Busca em profundidade   |
| 09 - Algoritmo de Dijkstra   |
| 10 - Gerar grafo aleatório    |
| 11 - Visualizar grafo        |
| -1 - Sair                    |
=====
Opção:
```

O usuário escolherá uma opção e os if's e else's farão o resto.

Grafo

1. Nosso coração do trabalho com certeza é esta classe. Nela está presente tudo o que precisa para funcionar perfeitamente.

```
1 from random import randint
2 import glob, os
3 import imageio
4 import pygraphviz as pgv
5
6
7 class Grafo:
8     def __init__(self):
9         self.vertices = []
10        self.matriz = [[0]*0 for i in range(0)]
11        self.predecessor = [None] * 0
12        self.time = 0
```

Primeiramente importamos várias libs que contribuíram bastante para o desenvolvimento deste projeto. Dentre elas a:

- `randint`: escolher números aleatórios para os pesos das arestas no método `gera_dijkstra()`.
- `glob` e `os`: são libs para comparação de `path names` e funcionalidades do sistema operacional, respectivamente.
- `imageio`: responsável pela criação dos `gifs` dos grafos.
- `pgv`: essa lib que nos proporcionou a criação visual dos grafos gerados.

Depois criamos a classe `Grafo` e chamamos o construtor `def __init__(self)` que serve para iniciarmos os atributos globais da nossa classe.

Métodos

Nós criamos vários métodos na classe `Grafo`. Detalharemos cada um a partir daqui.

Mostrar os Dados

Método simples que tem a função de mostrar a matriz (grafo) na tela.

```
1     def mostra_dados(self, matriz):
2         linha = coluna = len(matriz)
3         print()
4         for l in range(linha):
5             for c in range(coluna):
6                 print(f'\t{matriz[l][c]}', end='')
7             print()
```

Vértices Existentes

Retorna True se houver vértice na lista. A lista chama o método `count()` que retorna o número de ocorrências do valor passado por parâmetro. Se for igual a 0 significa que não há vértice no grafo.

```
1 def existe_vertice(self, v):  
2     return self.vertices.count(v) != 0
```

Inserir Vértice

Função para adicionar vértice ao grafo. Cria-se uma matriz auxiliar adicionando uma coluna e uma linha a mais e recebe cada posição da matriz principal. Retorna True se não houver nenhum vértice igual

```
1 def inserir_vertice(self, v):  
2     self.vertices.append(v)  
3     linha = coluna = len(self.matriz)  
4     matriz_aux = [[0]*(linha+1) for i in range(coluna+1)]  
5  
6     for l in range(linha):  
7         for c in range(coluna):  
8             matriz_aux[l][c] = self.matriz[l][c]  
9     self.matriz = matriz_aux
```

Inserir Aresta

Função para adicionar arestas num par de vértices. É atribuído o valor 1 na posição da matriz determinada por inicio e fim.

```
1 def inserir_aresta(self, inicio, fim):  
2     if not(self.existe_vertice(inicio)) or not(self.existe_vertice(  
3         fim)):  
4         return False  
5     if inicio == fim:  
6         print('\n\t\tApenas grafos simples. Tente novamente')  
7         return False  
8     else:  
9         self.matriz[self.vertices.index(inicio)][self.vertices.index(  
10             fim)] = 1  
11         self.matriz[self.vertices.index(fim)][self.vertices.index(  
12             inicio)] = 1  
13     return True
```

Checar se o Grafo é Completo

Checa se um grafo é completo ou não. Simplesmente verifica se há um 0 na metade triangular da matriz.

```
1 def checa_completo(self):
2     linha = coluna = len(self.matriz)
3
4     for l in range(linha):
5         for c in range(l+1, coluna):
6             if self.matriz[l][c] == 0:
7                 return False
8     return True
```

Grafo Complementar

Cria o grafo complementar da matriz. Percorre a matriz e verifica onde for 0, menos na diagonal principal, troca por 1, e onde for 1 troca por 0.

```
1 def complementar(self):
2     linha = coluna = len(self.matriz)
3     matriz_aux = [[0]*(linha) for i in range(coluna)]
4
5     for l in range(linha):
6         for c in range(coluna):
7             matriz_aux[l][c] = self.matriz[l][c]
8             if l != c and matriz_aux[l][c] == 1:
9                 matriz_aux[l][c] = 0
10            elif l != c:
11                matriz_aux[l][c] = 1
12    print()
13    self.mostra_dados(matriz_aux)
```

Gera Grafo Aleatório

Função para gerar um grafo aleatório com um tamanho específico.

```
1 def gera_aleatorio(self, tamanho):
2     matriz = [[0]*tamanho for coluna in range(tamanho)]
3
4     for i in range(tamanho):
5         for j in range(i+1, tamanho):
6             matriz[i][j] = randint(0,1)
7             matriz[j][i] = matriz[i][j]
8
9     linha = coluna = len(matriz)
10    print()
11    self.mostra_dados(matriz)
```

Número de Componentes

Função para contar o numero de componentes do grafo. A variável componentes recebe o valor dos predecessores da função de busca em largura e retorna a quantidade de None's encontrados.

```
1 def num_componentes(self):
2     componentes, _, _ = self.dfs()
3
4     return componentes.count(None)
```

Busca em Profundidade

Função retirada do conteúdo passado em sala.

```
1 def dfs(self):
2     linha = coluna = len(self.matriz)
3     cor = ['BRANCO'] * linha
4     self.predecessor = [None] * linha
5     d = [None] * len(self.matriz)
6     f = [None] * len(self.matriz)
7     for u in self.vertices:
8         if cor[u] in 'BRANCO':
9             self.dfs_util(u, cor, d, f)
10
11     self.time = 0
12
13     return self.predecessor, d, f
14
15 def dfs_util(self, u, cor, d, f):
16     linha = coluna = len(self.matriz)
17     cor[u] = 'CINZA'
18     self.time += 1
19     d[u] = self.time
20
21     for v, i in enumerate(self.matriz[u]):
22         if i == 1:
23             if cor[v] in 'BRANCO':
24                 self.predecessor[v] = u
25                 self.dfs_util(v, cor, d, f)
26     cor[u] = 'PRETO'
27     self.time += 1
28     f[u] = self.time
29
30     return self.predecessor, d, f, cor
31
```

Checa se o Grafo é uma Árvore

Função para verificar se o grafo é uma árvore. Primeiro verifica se há apenas 1 componente, se não já retorna False e depois verifica se há ciclo.

```
1 def checa_arvore(self):
2     if self.num_componentes() > 1:
3         return False
4     linha = len(self.matriz)
5     visitado = [False] * linha
6
7     for i in range(linha):
8         if visitado[i] == False:
9             if self.isCicle(i, visitado, -1):
10                 return False
11     return True
```

Algoritmo de Dijkstra

Cria o grafo baseado em pesos das arestas. Percorre toda a matriz e verifica, onde for 1 o valor é trocado por um peso aleatório de 1 a 10.

```
1 def dijkstra(self, inicio, fim):
2     if not(self.existe_vertice(inicio)) or not(self.existe_vertice(
3         fim)):
4         return False
5     matriz_dijkstra = self.gera_dijkstra()
6     self.mostra_dados(matriz_dijkstra)
7     linha = len(matriz_dijkstra)
8     distancia = [999] * linha
9     self.predecessor = [None] * linha
10    distancia[inicio] = 0
11    visitado = [False] * linha
12
13    for i in range(linha):
14        u = self.minDistancia(distancia, visitado)
15        visitado[u] = True
16        for v in range(linha):
17            if matriz_dijkstra[u][v] > 0 and visitado[v] == False and
18                \
19                distancia[v] > distancia[u] + matriz_dijkstra[u][v]:
20                distancia[v] = distancia[u] + matriz_dijkstra[u][v]
21                self.predecessor[v] = u
22    self.mostra_dijkstra(distancia, inicio, fim)
```

Visualizar Grafo

Função coringa do nosso trabalho. Conseguimos a partir da biblioteca pygraphviz criar um grafo e adicioná-lo a um arquivo de imagem para visualização. Primeiramente criamos as características do nosso grafo (resolução, cor de fundo, tipo de forma dos vértices, estilo e cor dos vértices) e depois adicionamos os vértices, as arestas, geramos o layout para poder esboçar o grafo.

Criamos um arquivo .dot que é o tipo de arquivo que se cria grafos com graphviz e então criamos a imagem para visualização, todos eles salvo no diretório Imagens na raiz do projeto.

```
1 def grafo_visual(self):
2     interface_grafo = pgv.AGraph()
3     interface_grafo.graph_attr['label'] = '\nTeoria dos Grafos\
4         \nMatheus Barbosa e Rafael Sidnei'
5     interface_grafo.graph_attr['dpi'] = 200
6     interface_grafo.graph_attr['bgcolor'] = 'mediumturquoise'
7     interface_grafo.node_attr['shape'] = 'circle'
8     interface_grafo.node_attr['style'] = 'filled'
9     interface_grafo.node_attr['fillcolor'] = 'white'
10    interface_grafo.node_attr['fontcolor'] = 'white'
11
12    linha = coluna = len(self.matriz)
13
14    if linha == 0:
15        return False
16
17    [interface_grafo.add_node(x, xlabel = f'V{x}') for x in range(
18        linha)]
19
20    for l in range(linha):
21        for c in range(l+1, coluna):
22            if self.matriz[l][c] == 1:
23                interface_grafo.add_edge(l, c)
24
25    interface_grafo.layout(prog='dot')
26    interface_grafo.write('/home/matheus/Documentos/Faculdade/5
27        Período Computação' +
28        '/Teoria dos Grafos/TrabalhoFinal/Imagens/grafo.dot')
29
30    interface_grafo.draw('/home/matheus/Documentos/Faculdade/5
31        Período Computação/' +
32        'Teoria dos Grafos/TrabalhoFinal/Imagens/grafo.png')
33
34    return interface_grafo
```


Busca em Largura

Com certeza essa foi a função mais impressionante que fizemos. Nela conseguimos (além executar o algoritmo de busca em largura perfeitamente) criar um algoritmo que gera gifs animados a partir do grafo que o próprio usuário montou. Ela faz todo o processo de coloração de vértices e arestas, além disso, mostra a distância necessária para chegar a cada vértice a partir do ponto de partida escolhido pelo usuário.

Utilizamos o algoritmo padrão mostrado em sala para executar a busca em largura. Já para criar os gifs utilizamos o suporte das libs já listadas anteriormente (pgv, glob, os, PIL e imageio).

```
1 def busca_largura(self, indice):
2     if not(self.existe_vertice(indice)):
3         return False
4
5     linha = len(self.matriz)
6     f = 10
7
8     # Criando grafo para executar a busca em largura
9     grafoV = self.grafo_visual()
10    node = grafoV.get_node(indice)
11    node.attr['fillcolor'] = 'gray'
12
13    for i in range(linha):
14        node = grafoV.get_node(i)
15        if i == indice:
16            node.attr['label'] = '0'
17        else:
18            node.attr['label'] = ' '
19            node.attr['fontcolor'] = 'crimson'
20
21    filename = f'grafo{f}.png'
22    grafoV.layout(prog='dot')
23
24    grafoV.draw('/home/matheus/Documents/Faculdade/5 Período
25               Computacao/' +
26               'Teoria dos Grafos/TrabalhoFinal/Imagens/' + filename)
27
28    cor = ['BRANCO'] * linha
29    distancia = [999] * linha
30    self.predecessor = [None] * linha
31
32    cor[indice] = 'CINZA'
33    distancia[indice] = 0
34    self.predecessor[indice] = None
35    fila = []
36    fila.append(indice)
```

```
36
37     while len(fila):
38         u = fila.pop(0)
39         node = grafoV.get_node(u)
40         f += 1
41
42         for v, i in enumerate(self.matriz[u]):
43             if i == 1:
44                 if cor[v] in 'BRANCO':
45                     cor[v] = 'CINZA'
46                     distancia[v] = distancia[u] + 1
47                     self.predecessor[v] = u
48                     fila.append(v)
49
50                     f += 1
51                     node = grafoV.get_node(v)
52                     edge = grafoV.get_edge(u, v)
53                     node.attr['fillcolor'] = 'gray'
54                     node.attr['label'] = f'{distancia[v]}'
55                     edge.attr['color'] = 'crimson'
56
57                     filename = f'grafo{f}.png'
58                     grafoV.layout(prog='dot')
59
60                     grafoV.draw('/home/matheus/Documentos/Faculdade/5
61                                 Periodo Computacao/' +
62                                 'Teoria dos Grafos/TrabalhoFinal/Imagens/' +
63                                 filename)
64
65                 f += 1
66                 cor[u] = 'PRETO'
67                 node = grafoV.get_node(u)
68                 node.attr['fillcolor'] = 'black'
69
70                 filename = f'grafo{f}.png'
71                 grafoV.layout(prog='dot')
72
73                 grafoV.draw('/home/matheus/Documentos/Faculdade/5
74                             Periodo
75                             Computacao/' +
76                             'Teoria dos Grafos/TrabalhoFinal/Imagens/' + filename)
77
78     print('\n\tPredecessores:\n')
79     for i in range(linha):
80         print(f'\tV{i}', end='')
81     print()
82     for pi in self.predecessor:
83         print(f'\t{pi}', end='')
```

```
81     print('\n')
82
83     path = ('/home/matheus/Documentos/Faculdade/5    Período
            Computacao/' +
84     'Teoria dos Grafos/TrabalhoFinal/Imagens/')
85
86     files = [f for f in glob.glob(path + '*.png')]
87
88     files.sort()
89     images = []
90     kargs = { 'duration': 2 }
91     for filename in files:
92         images.append(imageio.imread(filename))
93         os.remove(filename)
94
95     imageio.mimsave('/home/matheus/Documentos/Faculdade/5    Período
            Computacao/' +
96     'Teoria dos Grafos/TrabalhoFinal/Imagens/grafico.gif', images,
            'GIF', **kargs)
97
98     return True
```

Conclusão

Esse trabalho foi desafiador. Aprendemos coisas que nem sabíamos que existiam e que eram possíveis sem ser tão complexo assim. As bibliotecas usadas foram de enorme ajuda e sem elas não seria possível fazer nem 1/3 do que pretendíamos.

Referências

StackOverflow.com