Teoria dos Grafos

Documentação de Implementação – Projeto de Grafos (Parte 1)

Aluno	TIA
Amanda Laís Xavier Fontes	31949436
Thiago Henrique Quadrado Estacio	42012740
Rafael Junqueira Pezeiro	32035901

GitHub: https://github.com/Thiago2204/Projeto-Callisto

Apresentação:https://www.icloud.com/keynote/057uLVz98XDAUEwB896xOQnIw#Apresenta%C3%A7%C3%A3o

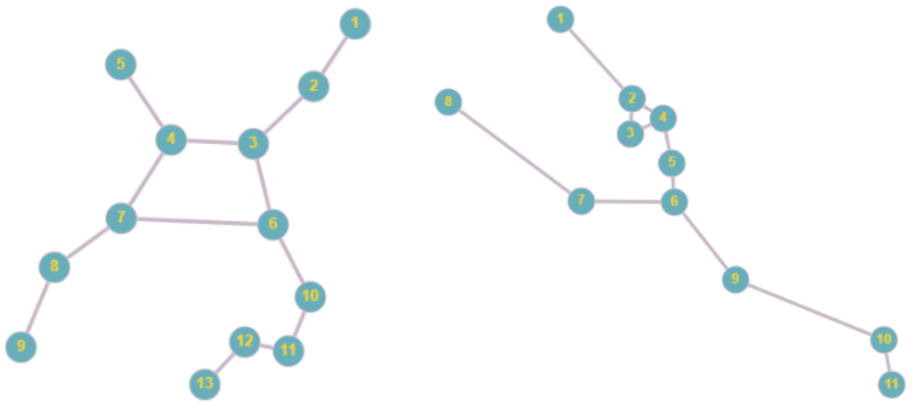
Replit: https://replit.com/join/kpysioikdj-thiagoestacio1

Descrição do Projeto:

O Projeto Callisto possui como objetivo analisar a quantidade de desenhos possíveis a partir de um número *n* fixo de estrelas em uma imagem do espaço.
O software recebe uma imagem do céu noturno e usa as estrelas nela contidas como vértices que serão unidas e, a partir desta ligação, serão criadas formas, ou seja, constelações.
Para a formação dessas constelações, algumas regras foram definidas:

Testagem do Projeto:

Testes Escolhidos



- Ler dados do arquivo grafo.txt:

Teste 1 	
Teste 2 	

- Gravar dados no arquivo grafo.txt:

Teste 1 	
Teste 2 	

- Inserir vértice:

Teste 1	

	
Teste 2 	

- Inserir aresta:

Teste 1 	
Teste 2 	

- Remove vértice:

Teste 1	

	
Teste 2 	

- Remove aresta:

Teste 1 	
Teste 2 	

- Mostrar conteúdo do arquivo:

Teste 1	

	
Teste 2 	

- Mostrar grafo:

Teste 1 	
Teste 2 	

- Encerrar a aplicação:

Teste 1	

	
Teste 2 	

Print do Grafo:

Total:
Número de Arestas = 118
Número de Vértices = 124

Código:

Main.py

```
from grafoMatriz import GrafoMatriz, TGrafoND
from grafoLista import GrafoLista
import math
import time
import os
```

```python
# GLOBALS ------------------------------------------------
NOME_ARQ = "grafo.txt"

# FUNÇÕES ------------------------------------------------
# Lê o arquivo txt e cria um grafo definido pelo seu tipo
# tipo -> se o grafo é orientado ou não
# t -> se for 0 é sem peso, 1 é com peso
# n -> quantidade de vértices
# m -> quantidade de arestas

def arq_grafo(n_aqr: str, tipo=0):
    # le as duas primeiras linhas para
    # definir o tipo (t) e a quantidade de vertices (n)
    # assim como quantidade de arestas (m)
    try:
        arq = open(n_aqr, 'r')
    except OSError:
        print("O arquivo informado não existe !!")
        return None
    t, n, m = int(arq.readline()), int(arq.readline()),
int(arq.readline())
    # Instancia o Grafo
    if tipo == 0 and t == 0:
        grafo = TGrafoND(n, False)
    elif tipo == 0 and t == 1:
        grafo = TGrafoND(n, True)
    elif tipo == 1 and t == 0:
        grafo = GrafoMatriz(n, False)
    else:
        grafo = GrafoMatriz(n, True)
    data = arq.readlines()
    arq.close()
    if t == 1:  # para os rotulados
        for linha in data:
            v, w, valor = linha.split()
            v, w, valor = int(v), int(w), int(valor)
            grafo.insere_a(v, w, valor)
    if t == 0:  # para não rotulados
        for linha in data:
            v, w = linha.split()
            v, w = int(v), int(w)
            grafo.insere_a(v, w)
    return grafo
```

```
def grafo_arq(grafo):
    arq = open("grafo.txt", 'w')
    #informacoes
    arq.write("1\n" + str(grafo.n) + "\n" + str(grafo.m) + "\n")
    temp = grafo.m
    for i in range(grafo.n):
        for x in range(grafo.n):
            if grafo.adj[i][x] != math.inf and x>i:
                #ligacoes
                arq.write(str(i) + " " + str(x) + " " +
str(grafo.adj[i][x]))
                temp -= 1
                if temp != 0:
                    arq.write("\n")
    arq.close()


def converter_ml(original: GrafoMatriz) -> GrafoLista:
    gl = GrafoLista(original.n)

    for v in range(0, original.n):
        for w in range(0, original.n):
            if original.adj[v][w] == 1:
                gl.insere_a(v, w)
    return gl


def saudacoes():
    print("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-")
    time.sleep(0.3)
    print("-*-*-*-*-*-*-*BEM VINDO*-*-*-*-*-*-*-*-*-*-*-*-*-*-")
    time.sleep(0.3)
    print("*-*-*-*-*-PROJETO CALLISTO*-*-*-*-*-*-*-*-*-*-*-*-*")
    time.sleep(0.3)
    print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
    time.sleep(0.3)
    print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
    time.sleep(0.3)
    print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
    time.sleep(0.3)
    print("\n")


def show_opcoes():
    print("|  --------------  Opções  ---------------- -")
    print("|  1) Ler dados de um arquivo txt           |")
    print("|  2) Gravar dados no arquivo txt           |")
    print("|  3) Inserir vértice                       |")
```

```
    print("|  4) Inserir aresta                          |")
    print("|  5) Remover vértice                         |")
    print("|  6) Remover aresta                          |")
    print("|  7) Mostrar conteúdo do arquivo             |")
    print("|  8) Mostrar grafo                           |")
    print("|  9) Encerrar a aplicação                    |")


def recebe() -> int:
    return int(input("| - Escolha uma das opções acima: "))


def falha():
    print("FALHA NA OPERAÇÃO!! - Grafo inexistente.")


def sucesso():
    print("SUCESSO NA OPERAÇÃO :D")


def op1():
    return str(input("Digite o nome do arquivo: "))


def op2(grafo=None):
    if not grafo:
        return False
    grafo_arq(grafo)
    return True


def op3(grafo=None):
    if not grafo:
        return False
    grafo.insere_v(grafo)
    return grafo


def op4(grafo=None):
    if not grafo:
        return False
    v = int(input("Informe o primeiro dos vértices que serão
interligados:\n"))
    w = int(input("Informe o segundo dos vértices que serão
interligados:\n"))
    if grafo.rotulado:
```

```
        p = float(input("Informe o custo da ligação (pode ser em ponto
flutuante): "))
        grafo.insere_a(v, w, p)
        return grafo
    grafo.insere_a(v, w)
    return grafo


def op5(grafo):
    if not grafo:
        return False
    v = int(input("Informe qual vértice será removido: "))
    grafo.remover(v)
    return grafo


def op6(grafo):
    if not grafo:
        return False
    v = int(input("Informe o primeiro vértice da ligação será removida:
"))
    w = int(input("Informe o segundo vértice da ligação será removida:
"))
    grafo.remove_a(v, w)
    return grafo


def op7():
    with open(op1()) as file:
        print(file.read())


def op8(grafo):
    if not grafo:
        return False
    grafo.show()


def menu():
    grafo = None
    saudacoes()
    while True:
        input("\n\n Precione qualquer tecla para continuar...")
        os.system('cls')
        show_opcoes()
        escolha = recebe()
```

```
        if escolha == 9:
            print("Até mais, estrelinha *-*")
            return True
        elif escolha == 1:
            grafo = arq_grafo(op1())
            if grafo:
                print("Grafo recuperado de um arquivo")
        elif escolha == 2:
            if not op2(grafo):
                falha()
                continue
            sucesso()
        elif escolha == 3:
            if not grafo:
                falha()
                continue
            grafo = op3(grafo)
            sucesso()
        elif escolha == 4:
            if not grafo:
                falha()
                continue
            grafo = op4(grafo)
        elif escolha == 5:
            if not grafo:
                falha()
                continue
            grafo = op5(grafo)
        elif escolha == 6:
            if not grafo:
                falha()
                continue
            grafo = op6(grafo)
        elif escolha == 7:
            op7()
        elif escolha == 8:
            if not grafo:
                falha()
                continue
            op8(grafo)

# MAIN ----------------------------------------------------
if __name__ == "__main__":
    menu()
```

grafoLista.py

```
# Grafo como uma lista de adjacência
class GrafoLista:
    TAM_MAX_DEFAULT = 100  # qtde de vértices máxima default

    # construtor da classe grafo
    def __init__(self, n=TAM_MAX_DEFAULT):
        self.n = n  # número de vértices
        self.m = 0  # número de arestas
        # lista de adjacência
        self.listaAdj = [[] for i in range(self.n)]

    # Insere uma aresta no Grafo tal que
    # v é adjacente a w
    def insere_a(self, v, w):
        self.listaAdj[v].append(w)
        self.m += 1

    # remove uma aresta v->w do Grafo
    def remove_a(self, v, w):
        self.listaAdj[v].remove(w)
        self.m -= 1

    # Apresenta o Grafo contendo
    # número de vértices, arestas
    # e a LISTA de adjacência obtida
    def show(self):
        print(f"\n n: {self.n:2d} ", end="")
        print(f"m: {self.m:2d}")
        for i in range(self.n):
            print(f"\n{i:2d}: ", end="")
            for w in range(len(self.listaAdj[i])):
                val = self.listaAdj[i][w]
                print(f"{val:2d}", end="")

        print("\n\nfim da impressao do grafo.")


##########################################################################
#######
    ##      Exercícios neste arquivo (grafoLista.py): 12, 14, 15, 16,
17, 19, 20 ##

##########################################################################
```

```
#######

    # EX12 --- se dois grafos direcionados sao iguais ||Amanda
    def compara(self, o):
        if self.listaAdj == o.listaAdj:
            print("Sao iguais")
            return True
        else:
            print("Nao sao iguais")
            return False

    # EX14 --- recebe um grafo em lista de adjacência e inverta a lista
de adjacência de todos os vértices || Amanda
    def inverte_lista(self, u):
        self.listaAdj[u].reverse()

    # EX15 - se é fonte ou não \\amanda -- usa os in e out degree pra
fazer esse
    def out_degree(self, v: int) -> int:
        return len(self.listaAdj[v])

    def in_degree(self, v: int) -> int:
        aparicoes = 0
        for vertice in self.listaAdj:
            for v2 in vertice:
                if v2 == v:
                    aparicoes += 1
        return aparicoes

    def is_fonte(self, v):
        if self.in_degree(v) == 0 and self.out_degree(v) > 0:
            return 1
        return 0

    # EX16
    def is_sorvedouro(self, v: int) -> int:
        if self.in_degree(v) > 0 and self.out_degree(v) == 0:
            return 1
        return 0

    # EX17 --- receba um grafo dirigido e retorna 1 se o grafo for
simétrico ou 0 || Amanda
    def is_symm(self):
        j = 0
        for i in self.listaAdj:
            if len(i) == 0:
```

```
                continue
            for w in i:
                if j not in self.listaAdj[w]:
                    return 0
            j += 1
        return 1

    # EX19 --- remover vértices de um grafo direcionado e não
direcionado
    def remover(self, v: int) -> int:
        if v < self.n:
            for vertice in range(len(self.listaAdj) - 1):
                if v in self.listaAdj[vertice]:
                    self.remove_a(vertice, v)
                if v == vertice:
                    for a in self.listaAdj[v]:
                        self.remove_a(v, a)
            del self.listaAdj[v]
            self.n -= 1
            return 1
        return 0

    # EX20
    def completo(self, v):
        checa = 1
        for i in range(self.n):
            if v in self.listaAdj[i] == 1:
                continue
            else:
                checa = 0
                break
        if checa == 1:
            return 1
        else:
            return 0
```

grafoMatriz.py

```
import math
```

```python
# CLASSES
# grafo matriz  --  rotulado ou não
# grafo nao direcionado -- rotulado ou não
# Implementar uma classe Vértice

class GrafoMatriz:
    TAM_MAX_DEFAULT = 100  # qtde de vértices máxima default
    # construtor da classe grafo
    def __init__(self, n=TAM_MAX_DEFAULT, rotulado=False):
        self.n = n  # número de vértices
        self.m = 0  # número de arestas
        self.rotulado = False  # servirá de verificação nos métodos
abaixo
        # matriz de adjacência
        if rotulado:
            self.rotulado = True
            self.adj: list[list[float]] = [[math.inf for i in range(n)]
for j in range(n)]
        else:
            self.adj = [[0 for i in range(n)] for j in range(n)]

    # Insere uma aresta no Grafo tal que
    # v é adjacente a w
    def insere_a(self, v, w, rotulo=-1):  # rotulo -1 é para quando não
se sabe o peso daquela aresta
        if self.rotulado and self.adj[v][w] == math.inf:
            self.adj[v][w] = rotulo
            self.m += 1
        if not self.rotulado and self.adj[v][w] == 0:
            self.adj[v][w] = 1
            self.m += 1  # atualiza qtd arestas

    def insere_v(self):
        self.n += 1
        if self.rotulado:
            for linha in self.adj:
                linha.append(math.inf)
            self.adj.append([math.inf for i in range(self.n)])
        else:
            self.adj.append([0 for i in range(self.n)])

    # remove uma aresta v->w do Grafo
    def remove_a(self, v, w):
        # testa se temos a aresta
        if not self.rotulado and self.adj[v][w] == 1:
            self.adj[v][w] = 0
```

```
            self.m -= 1  # atualiza qtd arestas
        if self.rotulado and self.adj[v][w] != math.inf:
            self.adj[v][w] = math.inf
            self.m -= 1

    # Apresenta o Grafo contendo
    # número de vértices, arestas
    # e a matriz de adjacência obtida
    def show(self):
        print(f"\n n: {self.n:2d} m: {self.m:2d} r: {self.rotulado}\n")
        for i in range(self.n):
            for w in range(self.n):
                print(f"Adj[{i:2d},{w:2d}] = {self.adj[i][w]} | ",
end="")
            print("\n")
        print("\nfim da impressao do grafo.")

    # Apresenta o Grafo contendo
    # número de vértices, arestas
    # e a matriz de adjacência obtida
    # Apresentando apenas os valores 0 ou 1
    def show_min(self):
        print(f"\n n: {self.n:2d} m: {self.m:2d} r: {self.rotulado}\n")
        for i in range(self.n):
            for w in range(self.n):
                print(f" {self.adj[i][w]} |", end="")
            print("\n")
        print("\nfim da impressao do grafo.")

    def in_degree(self, v: int) -> int:
        return len([linha for linha in self.adj if linha[v] != 0 and
linha[v] != math.inf])

    def out_degree(self, v: int) -> int:
        return len([sai for sai in self.adj[v] if sai != 0 and sai !=
math.inf])

    def is_fonte(self, v: int) -> int:
        if self.in_degree(v) == 0 and self.out_degree(v) > 0:
            return 1
        return 0

    def is_sorvedouro(self, v) -> int:
        if self.in_degree(v) > 0 and self.out_degree(v) == 0:
            return 1
        return 0
```

```
    def is_simetrico(self) -> int:
        for i in range(len(self.adj)):
            for j in range(len(self.adj[i])):
                if self.adj[i][j] != self.adj[j][i]:
                    return 0
                else:
                    return 1

    def remover(self, v: int) -> int:
        if v < self.n:
            # Remove as arestas
            for _ in range(0, len(self.adj[v])):
                self.remove_a(v, _)
                self.remove_a(_, v)
            # Remove os vértices
            for linha in self.adj:
                del linha[v]
            del self.adj[v]
            self.n -= 1
            return 1
        else:
            return 0

    def completo(self) -> int:  # dei ctrl c ctrl v
        checa = 1
        for i in range(self.n):
            for w in range(self.n):
                if i != w:
                    if self.adj[i][w] != 0 and self.adj[i][w] !=
math.inf:
                        continue
                    else:
                        checa = 0
                        break
        if checa == 1:
            return 1
        else:
            return 0

    @staticmethod
    def visitar_no(no: int):
        print(f"Estamos visitando o nó {no}")

    @staticmethod
    def marcar_no(marcados, no):
```

```
        marcados.append(no)
        return marcados

    def no_adjacente(self, no, marcados):
        adjs = self.adj[no]
        for _ in range(self.n):
            if (adjs[_] != 0 and adjs[_] != math.inf) and _ not in
marcados:
                return _
        return -1

    def nos_adjacentes(self, no, marcados):
        return [index for index, valor in enumerate(self.adj[no]) if
(valor != 0 and valor != math.inf) and marcados]

    def percurso_profundidade(self, v_inicio):
        marcados = []
        p = Pilha()
        self.visitar_no(v_inicio)
        marcados = self.marcar_no(marcados, v_inicio)
        p.push(v_inicio)
        while not p.is_empty():
            no_atual = p.pop()
            no_seguinte = self.no_adjacente(no_atual, marcados)
            while no_seguinte != -1:
                self.visitar_no(no_seguinte)
                p.push(no_atual)
                self.marcar_no(marcados, no_seguinte)
                no_atual = no_seguinte
                no_seguinte = self.no_adjacente(no_seguinte, marcados)



class TGrafoND:
    TAM_MAX_DEFAULT = 100
    def __init__(self, n=TAM_MAX_DEFAULT, rotulado=False):
        self.n = n
        self.m = 0
        self.rotulado = False

        if rotulado:
            self.rotulado = True
            self.adj = [[math.inf for i in range(n)] for j in range(n)]
        else:
            self.adj = [[0 for i in range(n)] for j in range(n)]
```

```
    def insere_v(self):
        self.n += 1
        if self.rotulado:
            for linha in self.adj:
                linha.append(math.inf)
            self.adj.append([math.inf for i in range(self.n)])
        else:
            self.adj.append([0 for i in range(self.n)])

    def insere_a(self, v, w, valor: float = 1):
        if self.rotulado and self.adj[v][w] == math.inf:
            self.adj[v][w], self.adj[w][v] = valor, valor
            self.m += 1
        if not self.rotulado and self.adj[v][w] == 0:
            self.adj[v][w], self.adj[w][v] = valor, valor
            self.m += 1

    def remove_a(self, v, w):
        if self.rotulado and self.adj[v][w] != math.inf:
            self.adj[v][w], self.adj[w][v] = math.inf, math.inf
            self.m -= 1
        if not self.rotulado and self.adj[v][w] != 0:
            self.adj[v][w], self.adj[w][v] = 0, 0
            self.m -= 1

    def show(self):
        if self.rotulado:
            print(f"\n n: {self.n:2d} ", end="")
            print(f"m: {self.m:2d}\n")
            for i in range(self.n):
                for w in range(self.n):
                    if self.adj[i][w] != math.inf:
                        print(f"Adj[{i:2d},{w:2d}] = ", self.adj[i][w],
end=" ")
                    else:
                        print(f"Adj[{i:2d},{w:2d}] = 0 ", end="")
                print("\n")
            print("\nfim da impressao do grafo.")
        else:
            print(f"\n n: {self.n:2d} ", end="")
            print(f"m: {self.m:2d}\n")
            for i in range(self.n):
                for w in range(self.n):
                    if self.adj[i][w] == 1:
                        print(f"Adj[{i:2d},{w:2d}] = 1 ", end="")
                    else:
```

```
                        print(f"Adj[{i:2d},{w:2d}] = 0 ", end="")
                print("\n")
            print("\nfim da impressao do grafo.")

    def show_min(self):
        print(f"\n n: {self.n:2d} ", end="")
        print(f"m: {self.m:2d}\n")
        for i in range(self.n):
            for w in range(self.n):
                if self.rotulado:
                    if self.adj[i][w] != math.inf:
                        print(" ", self.adj[i][w], end=" ")
                    else:
                        print(" 0 ", end="")
                else:
                    if self.adj[i][w] == 1:
                        print(" 1 ", end="")
                    else:
                        print(" 0 ", end="")
            print("\n")
        print("\nfim da impressao do grafo.")

    def in_degree(self, v: int) -> int:
        return len([linha for linha in self.adj if linha[v] != 0 and
linha[v] != math.inf])

    def out_degree(self, v: int) -> int:
        return len([sai for sai in self.adj[v] if sai != 0 and sai !=
math.inf])

    def is_fonte(self, v: int) -> int:
        if self.in_degree(v) == 0 and self.out_degree(v) > 0:
            return 1
        return 0

    def is_sorvedouro(self, v: int) -> int:
        if self.in_degree(v) > 0 and self.out_degree(v) == 0:
            return 1
        return 0

    @staticmethod
    def is_simetrico() -> int:
        return 1

    def remover(self, v: int) -> int:
        if v < self.n:
```

```
            # Remove as arestas
            for _ in range(0, len(self.adj[v])):
                self.remove_a(v, _)
                self.remove_a(_, v)
            # Remove os vértices
            for linha in self.adj:
                del linha[v]
            del self.adj[v]
            self.n -= 1
            return 1
        else:
            return 0

    def completo(self) -> int:  # dei ctrl c ctrl v
        checa = 1
        for i in range(self.n):
            for w in range(self.n):
                if i != w:
                    if self.adj[i][w] != 0 and self.adj[i][w] !=
math.inf:
                        continue
                    else:
                        checa = 0
                        break
        if checa == 1:
            return 1
        else:
            return 0
```