

Teoria dos Grafos

Documentação de Implementação – Projeto de Grafos (Parte 2)

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

Conteúdo do Relatório

GitHub:

<https://github.com/Thiago2204/Projeto-Callisto>

Apresentação:

<https://www.icloud.com/keynote/057uLVz98XDAUEwB896xOQnlw#Apresenta%C3%A7%C3%A3o>

Replit:

<https://replit.com/join/jwbokpuvpb-amandalais>

Descrição do Projeto:

O Objetivo do Projeto consiste em criar uma rota de para que num futuro distante a raça humana possa atravessar a galáxia.

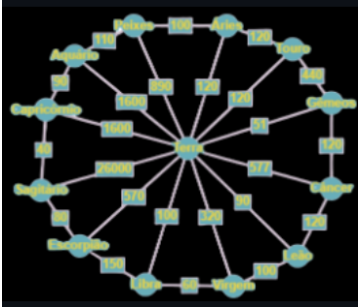

Esse projeto tem o objetivo de satisfazer os quesitos 4 e 9 da ODS:

- **ODS 9: Inovação infraestrutura – Construir infraestrutura resiliente, promover a industrialização inclusiva e sustentável, e fomentar a inovação.**
- Nosso projeto atende os objetivos de infraestrutura resiliente e promove a industrialização inclusiva e sustentável. O quesito de infraestrutura é contemplado pela construção de meios de transporte que sejam capazes de realizar o percurso definido pelo nosso projeto, graças ao seu potencial de obrigar a indústria espacial a construir máquinas capazes de fazerem tais percursos atualmente impossíveis. Já na parte de infraestrutura, ao serem criadas essas máquinas atualmente inexistentes.
- **ODS 4: Educação de qualidade – Assegurar a educação inclusiva, equitativa e de qualidade, e promover oportunidades de aprendizagem ao longo da vida para todos.**
- Propomos que as massas tenham uma melhor educação sobre os Cosmos, tendo um maior interesse pelas estrelas para que possamos criar novas gerações mais capacitadas e interessadas no assunto para que possa haver mais pesquisas no futuro, podendo, até, criar mais projetos que impulsionam a humanidade no futuro. Além disso, envia-se a identificação de padrões nas características de constelações,

de maneira a aprender sobre os jeitos que as constelações foram definidas pelas culturas.

Testagem do Projeto:

Caminho Mínimo:

Input	Output
 <pre> 0, 120, 440, 51, 577, 90, 320, 100, 570, 26000, 1600, 1600, 890, 120, 0, 120, 0, 0, 0, 0, 0, 0, 0, 0, 100, 440, 120, 0, 440, 0, 0, 0, 0, 0, 0, 0, 0, 51, 0, 440, 0, 120, 0, 0, 0, 0, 0, 0, 0, 577, 0, 0, 120, 0, 120, 0, 0, 0, 0, 0, 0, 90, 0, 0, 0, 120, 0, 100, 0, 0, 0, 0, 0, 320, 0, 0, 0, 0, 100, 0, 60, 0, 0, 0, 0, 100, 0, 0, 0, 0, 0, 60, 0, 150, 0, 0, 0, 570, 0, 0, 0, 0, 0, 0, 150, 0, 80, 0, 0, 26000, 0, 0, 0, 0, 0, 0, 0, 80, 0, 40, 0, 1600, 0, 0, 0, 0, 0, 0, 0, 0, 40, 0, 40, 1600, 0, 0, 0, 0, 0, 0, 0, 0, 40, 0, 110, 890, 100, 0, 0, 0, 0, 0, 0, 0, 0, 110, 0, </pre>	 <pre> - Escolha uma das opções acima: 10 {0: 0, 1: 120, 2: 240, 3: 51, 4: 171, 5: 90, 6: 150, 7: 100, 8: 180, 9: 220, 10: 260, 11: 220} </pre>

Conexidade:

Input	Output
<pre> 1 4 4 0 1 1 1 3 1 2 3 1 </pre>	<pre> Adj[0, 0] = 0 Adj[0, 1] = 1 Adj[0, 2] = 0 Adj[0, 3] = 0 Adj[1, 0] = 1 Adj[1, 1] = 0 Adj[1, 2] = 0 Adj[1, 3] = 1 Adj[2, 0] = 0 Adj[2, 1] = 0 Adj[2, 2] = 0 Adj[2, 3] = 1 Adj[3, 0] = 0 Adj[3, 1] = 1 Adj[3, 2] = 1 Adj[3, 3] = 0 fim da impressao do grafo. Partindo de TERRA, * :. . * *°C☆. . * :. . * * :. * . * . * :. Agora em ARIES *°C☆. . * :. . * * :. * . * . * :. Agora em GEMEOS *°C☆. . * :. . * * :. * . * . * :. Agora em TOURO Percurso CONCLUÍDO -.- * -.- É CONEXO -.- * -.- </pre>

Código

Main

```
from grafoMatriz import TGrafoND
import time
import os
import math

# 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)
    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')
    arq.write("1\n")
    arq.write(str(grafo.n)+ '\n') #vértices
    arq.write(str(grafo.m)+ '\n') #arestas
    for i in range(grafo.n):
        for x in range(grafo.n):
            if grafo.adj[i][x] != math.inf:
                arq.write(str(i) + ' ' + str(x) + ' ' +
str(grafo.adj[i][x]) + '\n')
    arq.close()

```

```

def saudacoes():
    print("
    .★..")
    print(" |Projeto Callisto| ")
    print(" ..★. ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    print("
    ")
    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) Verificar menor caminho |")
print("| 10) Verificar conexidade |")
print("| 11) Visualizar um percurso para a rota |")
print("| 12) 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 op9(grafo):
    if not grafo:
        return False
    grafo.dijkstra(grafo, 0)

def op10(grafo):
    if not grafo:
        return False
    v = int(input("Digite o número de um vértice: "))
    print(grafo.conexidade(grafo, v))

def op11(grafo):

```

```

if not grafo:
    return False
v = int(input("Digite o número do vértice inicial: "))
perc = grafo.percurso_profundidade(v)
grafo.percurso_feito(perc)

def menu():
    grafo = None
    saudacoes()
    while True:
        input("\n\n Precione qualquer tecla para continuar...")
        os.system('cls')
        show_opcoes()
        escolha = recebe()
        if escolha == 12:
            print(" * . * . * -` ★ `(-")
            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)
    elif escolha == 9:
        if not grafo:
            falha()
            continue
        op9(grafo)
    elif escolha == 10:
        if not grafo:
            falha()
            continue
        op10(grafo)
    elif escolha == 11:
        if not grafo:
            falha()
            continue
        op11(grafo)

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

```

grafoMatriz

```

import math
from util import Pilha
from queue import PriorityQueue

# CLASSES
# grafo nao direcionado -- rotulado ou não

class TGrafoND:
    TAM_MAX_DEFAULT = 100
    def __init__(self, n=TAM_MAX_DEFAULT, rotulado=False):
        self.n = n #vertices
        self.m = 0 #arestas
        self.rotulado = False
        self.visitados = []
        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

    @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 is_adjac(self, i, j) -> int:
    if (self.adj[i][j] != 0 or self.adj[j][i] != 0) == True:
        return True

def is_adjacto(self, i):
    vertices = []
    for x in range(self.n):
        if self.adj[i][x] or self.adj[x][i] > 0:
            vertices.append(x)
    return(vertices)

def percurso_profundidade(self, v_inicio):
    marcados = []
    visita = []
    p = Pilha()
    visita.append(v_inicio)
    marcados = self.marcas_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:
            visita.append(no_seguinte)
            p.push(no_atual)
            self.marcas_no(marcados, no_seguinte)
            no_atual = no_seguinte
            no_seguinte = self.no_adjacente(no_seguinte, marcados)
    return visita

def dijkstra(self, grafo, no_inicial):
    D = {n:float('inf') for n in range(grafo.n)}
    D[no_inicial] = 0
    pq = PriorityQueue()
    pq.put((0, no_inicial))
    while not pq.empty():
        (dist, no_atual) = pq.get()
        grafo.adj.append(no_atual)
        for vizinho in range(grafo.n):
            if grafo.adj[no_atual][vizinho] != -1:
                dist = grafo.adj[no_atual][vizinho]
                if vizinho not in grafo.visitados:
                    custo_antigo = D[vizinho]

```

```

        novo_custo = D[no_atual] + dist
        if novo_custo < custo_antigo:
            pq.put((novo_custo, vizinho))
            D[vizinho] = novo_custo

    print(D)

def conexidade(self, grafo, no_inicial):
    if len(self.percurso_profundidade(no_inicial)) == self.n:
        return "\nÉ CONEXO -; ★ '-"
    else:
        return "\nÉ DESCONEXO -; ★ '-"

def constelacao(self, num_const):
    match num_const:
        case 0:
            return "TERRA"
        case 1:
            return "ARIES"
        case 2:
            return "TOURO"
        case 3:
            return "GEMEOS"
        case 4:
            return "CANCER"
        case 5:
            return "LEAO"
        case 6:
            return "VIRGEM"
        case 7:
            return "LIBRA"
        case 8:
            return "ESCORPIAO"
        case 9:
            return "SAGITARIO"
        case 10:
            return "CAPRICORNIO"
        case 11:
            return "AQUARIO"
        case 12:
            return "PEIXES"

def percurso_feito(self, num_const):
    for i in num_const:
        msg = ""
        if i == 0:
            msg += "Partindo de "

```

```

        else:
            print(" ★ ° ☾ ☆ ,. , ★ :. . • , * :. \n")
            msg += ", * . • , . • , :. Agora em "
            msg += self.constelacao(i)
            if i == 0:
                msg += ", * :. . • ,"
            print(msg)
        print("\n\nPercurso CONCLUÍDO -` ★ `-'")

```

Util

```

class Pilha:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len(self.items)

class Fila:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
    def size(self):
        return len(self.items)

```