Caio Uehara Martins

nUSP 13672022

DCM - FFCLRP

Professor: EVANDROE.S.RUIZ

Trabalho 2 - Algoritmos e Estruturas de Dados II

- Google collab: <https://colab.research.google.com/drive/130d9QKNnLY19eVkCdqeUxn6jFD57rjMH?usp=sharing>

- O projeto usa o dataset do quakers dentro de uma pasta chamada “data” para instalar o quakers dataset, segue o link

<https://github.com/melaniewalsh/sample-social-network-datasets/tree/26249c33801b445b625ef85b1706854ff45aa1a3/sample-datasets/quakers>

**1. Importação de módulos**

#bibliotecas de sistema

import os

#limpeza de tela

from IPython.display import clear\_output

#bibilioteca gráfica para grafos

import networkx as nx

import matplotlib.pyplot as plt

#biblioteca de estruturação de dados

import pandas as pd

#Número aleatório

import random

**2. Treinando um pouco com networkx**

#explorando o networkx, criando um grafo aleatório e uma árvore de busca profunda

#criando grafo de uma rede social fictícia para uso dos algoritmos

G = nx.Graph()

# Adicionando nós (pessoas) ao grafo

lista\_nomes = ["Alice", "Bob", "Charlie", "David", "Eve","Volya","Sorokin","King","Pico","Ivanoff","Stepanida","Petroff","Bobko","Sergius","Berezin","Aktyx",

               "Nicodemus","Bosmuyrium","Aktyx","Y","Amaliya","Cloudian","Sim","Pavlov","Valentina","Inca","Pudens","Evgeni"]

G.add\_nodes\_from(lista\_nomes)

# Adicionando relações (arestas) entre as pessoas

def createRandomEdges(*n*):

    tamanho = len(lista\_nomes)

    for i in range(0, *n*):

        numero\_aleatorio = random.randrange(0, tamanho - 1)

        numero\_aleatorio2 = random.randrange(0, tamanho - 1)

        if(numero\_aleatorio != numero\_aleatorio2):

            G.add\_edge(lista\_nomes[numero\_aleatorio], lista\_nomes[numero\_aleatorio2])

createRandomEdges(50)

# Desenhando o grafo

def draw(*graph*, \**args*):

    common\_params = {

        "with\_labels": True,

        "font\_weight": 'bold',

        "node\_size": 20,

        "node\_color": 'skyblue',

        "font\_size": 8,

        "font\_color": 'black',

        "arrowsize": 10,

        "arrowstyle": '->'

    }

    if *args*:

        nx.draw(*graph*, *args*[0], \*\*common\_params)

    else:

        nx.draw(*graph*, \*\*common\_params)

    #Exibe o grafo

    plt.show()

pos = nx.spring\_layout(G)  # Posicionamento dos nós

draw(G, pos)

#Árvore DFS

DFS = nx.dfs\_tree(G, "Pudens", 5)

draw(DFS)

**3. Carregando dataset quakers**

#carregando os CSV com os dados da rede "Quakers"

quakers\_nodelist = pd.read\_csv(os.path.relpath("../data/quakers\_nodelist.csv"))

quakers\_edgelist = pd.read\_csv(os.path.relpath("../data/quakers\_edgelist.csv"))

display(quakers\_nodelist)

display(quakers\_edgelist)

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**4. Criando o grafo da rede**

#criando o grafo da rede

Q = nx.Graph()

#criando os nós e as arestas

for i in quakers\_nodelist.itertuples(*index*=False, *name*=None):

    name, \*attr = i

    Q.add\_node(name,

*Historical\_Significance*=attr[0],

*Gender*=attr[1],

*Birthdate*=attr[2],

*Deathdate*=attr[3],

*num*=0, #contador para o percurso

    )

Q.add\_edges\_from(list(quakers\_edgelist.itertuples(*index*=False, *name*=None)))

display(nx.nodes(Q))

display(nx.edges(Q))

display(Q.nodes["Joseph Wyeth"])

Texto

Descrição gerada automaticamente

**4. Algoritmo DFS**

#algoritmo DFS (depthFirstSearch) baseado no ppt da aula

"""

Um comentário sobre a implementação do algoritmo.

Ela foi realizada acessando a estrutura do grafo do networkx, para

assim poder ser explorado seus recursos. Assim, a manipulação das estruturas está

supondo trabalhar com o grafo da biblioteca e não uma lista, de forma que é necessário

criar o grafo primeiro.

"""

"""

Algoritmo em pseudocódigo

DFS(v)

    num(v) = i

    i = i + 1

    for todos\_vertices u adjacentes a v

        if num(u) == 0

            anexa\_aresta(uv) a edges

            DFS(u)

depthFirstSearch(G)

    for todos\_vertices v em G

        num(v) == 0

    edges = NULL; \\lista de arestas

    i=1;

    while(existe(v) tq. num(v)==0)

        DFS(v)

    return(edges)

"""

def resetNum(*graph*):

    for node\_name in *graph*.nodes():

        node = *graph*.nodes[node\_name]

        node["num"] = 0*;*

def DFS(*v\_name*, *Q*, *edges*):

    v = *Q*.nodes[*v\_name*]

    v["num"] +=1

Foto em preto e branco com texto preto sobre fundo branco

Descrição gerada automaticamente    for u\_name in *Q*.neighbors(*v\_name*): #percorre e encontra os adjacentes do nó dado (v)

        u = *Q*.nodes[u\_name]  # todo vértice u que é adjacente a v

        if u["num"] == 0:

            DFS(u\_name, *Q*, *edges*)

*edges*.append( (u\_name, *v\_name*) ) #anexa aresta

def depthFirstSearch(*graph*):

    resetNum(*graph*)*;*

    edges = list()

    for v\_name in *graph*.nodes():

        v = *graph*.nodes[v\_name]

        if v["num"] == 0:

            DFS(v\_name, *graph*, edges)

    for k\_name in *graph*.nodes:

        k = *graph*.nodes[k\_name]

        k["num"] = 0*;*

        DFS(k\_name, *graph*, edges)

    return edges*;*

depthFirstSearch(Q)

**5. Algoritmo DFS com colaração e função draw**

#draw loop function

pos = nx.spring\_layout(Q, *k*=0.5) #set posições fixas do grafo

def draw(*graph*):

    clear\_output(*wait*=True)

    plt.figure(*figsize*=(60,30))

    color\_map = list(nx.get\_node\_attributes(Q, "cor").values())

    """

    Notação formal das cores:

        brancos -> Vértice ainda não visitado

        cinza -> Vértice descoberto

        pretos -> Vértice já visitado. Terminado

    Notação transormada para melhor visualização:

        branco -> Vermelho

        cinza -> Cinza

        pretos -> Preto

    """

    def color\_transform(*color*):

        match *color*:

            case "cinza":

                return (239/255, 198/255, 170/255)

            case "branco":

                return (255/255, 0/255, 0/255)

            case "preto":

                return (0/255, 0/255, 0/255)

            case \_:

                return (255/255, 255/255, 255/255)

    color\_map = list(map(color\_transform , color\_map))

    params = {

        "with\_labels": True,

        "font\_weight": 'bold',

        "node\_color": color\_map,

        "node\_size": 1000,

        "font\_size": 6,

        "font\_color": 'black',

        "arrowsize": 5,

        "arrowstyle": '->'

    }

    nx.draw(*graph*, pos, \*\*params)

    #Exibe o grafo

    plt.show()

#versão realizada pelo LLM (ChatGPT) para adicionar o atributo cores

def add\_colors(*graph*):

    colors = {}  # Dicionário para armazenar cores associadas a cada nó

    for node\_name in *graph*.nodes():

        colors[node\_name] = 'branco'  # Inicialmente, todos os nós são brancos

    nx.set\_node\_attributes(*graph*, colors, 'cor')

def reset\_attributes(*graph*):

    for v in *graph*.nodes():

*graph*.nodes[v]['num'] = 0

*graph*.nodes[v]['cor'] = 'branco'

def DFS(*v\_name*, *Q*, *edges*):

    v = *Q*.nodes[*v\_name*]

    v['num'] += 1

    v['cor'] = 'cinza'

    for u\_name in *Q*.neighbors(*v\_name*):

        u = *Q*.nodes[u\_name]

        if u['cor'] == 'branco':

*edges*.append((u\_name, *v\_name*))

            draw(*Q*)

            DFS(u\_name, *Q*, *edges*)

    v['cor'] = 'preto'

def depthFirstSearch(*graph*):

    reset\_attributes(*graph*)

    edges = list()

    for v\_name in *graph*.nodes():

        v = *graph*.nodes[v\_name]

        if v['cor'] == 'branco':

            DFS(v\_name, *graph*, edges)

    reset\_attributes(*graph*)

    return edges

#Exemplo de uso no Quakers

add\_colors(Q)  # Adiciona o atributo 'cor' a cada nó

result = depthFirstSearch(Q)

print(result)

\*Esse parte do código renderiza o grafo em loop, mostrando sua coloração, os exemplos colocados aqui foram tirados em 3 estados diferentes (início, intermediário e final)

\*O padrão de coloração foi alterado para melhor visualização do grafo, como comentado no código

    Notação transormada para melhor visualização:

        branco -> Vermelho

        cinza -> Cinza

        pretos -> Preto

Estado inicial:

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

Estado intermediário:

Pipa no céu

Descrição gerada automaticamente com confiança baixa

Estado final:

Mapa com linhas pretas em fundo branco

Descrição gerada automaticamente com confiança média