



UFRJ
UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

Rede de fluxos em grafos direcionados

TRABALHO PRÁTICO 3

Trabalho apresentado à disciplina de Teoria dos Grafos da Universidade Federal do Rio de Janeiro (UFRJ), como exigência parcial do curso de Engenharia de Computação e Informação.

Orientação: Prof. Dr. Daniel Ratton

Daniel Rodrigues Ferreira e
Lucas Garcia Santiago de Abreu

SUMÁRIO

1

Lista de Adjacência

2

Arestas

3

Ford-Fulkerson

4

Caminhos

5

BFS

6

Gargalos

7

Alocação de Fluxo

8

Resultados



UFRJ

UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

Lista de Adjacência

- IMPLEMENTAÇÃO COM DICIONÁRIO
- VELOCIDADE DE LEITURA
- FACILIDADE DE MANIPULAÇÃO

```
class Grafo:
    def __init__(self, dados:str, direcionado: bool = True) -> None:
        self.dados = dados

        self.grafo = defaultdict(list)
        arquivo = open(self.dados, 'r')
        self.tamanho = int(arquivo.readline())

        for linha in arquivo: #Criando grafo residual
            original = self.add_aresta(vertice1, vertice2, capacidade, 0, False)
            reversa = self.add_aresta(vertice2, vertice1, capacidade, 0, True)

            original.reversa_pointer = reversa
            reversa.original_pointer = original

        def add_aresta(self, vertice1: int, vertice2: int, capacidade:int, fluxo:int, reversa:bool) -> None:
            self.grafo[vertice1].append(Aresta(vertice1, vertice2, capacidade, fluxo, reversa))

        def get_vizinhos(self, vertice: int):
            vizinhos = self.grafo[vertice]
            return vizinhos
```

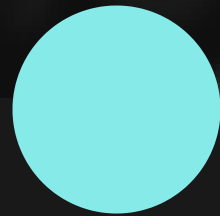
Arestas

- CAPACIDADE E FLUXO
- ARESTAS REVERSAS
- GRAFO RESIDUAL IMBUTIDO

```
class Aresta:
    def __init__(self, vertice1: int, vertice2:int, capacidade:int, fluxo:int, reversa:bool) -> None:
        """0 vertice1 aponta para o vertice2, por meio de uma aresta com capacidade e fluxo
        """
        if reversa:
            self.capacidade_residual = self.fluxo
        else:
            self.capacidade_residual = self.capacidade - self.fluxo

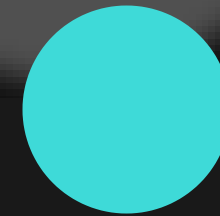
    def atualizar(self, gargalo:int):
        self.fluxo += gargalo
        if self.reversa == True:
            self.capacidade_residual = self.fluxo
        else:
            self.capacidade_residual = self.capacidade - self.fluxo
            self.reversa_pointer.atualizar(gargalo)
```


Ford-Fulkerson



Algoritmo de Ford-Fulkerson

```
if delta == True:
    capacidade = self.get_capacidade(inicio)
    delta = ceil(capacidade/2)
    while delta != 1:
        caminho = self.get_caminho(inicio, fim, delta)
        while caminho:
            self.aumentar(caminho)
            caminho = self.get_caminho(inicio, fim, delta)
        delta = ceil(delta/2)
    caminho = self.get_caminho(inicio, fim)
    while caminho:
        self.aumentar(caminho)
        caminho = self.get_caminho(inicio, fim)
```

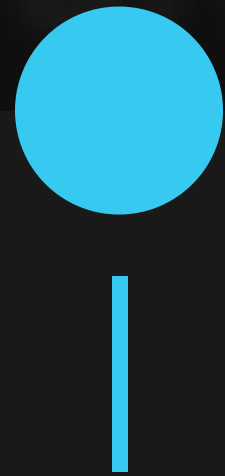


Obtenção do Fluxo

```
def get_fluxo(self, vertice: int) -> int:
    fluxo = 0
    vizinhos = self.get_vizinhos(vertice)
    for aresta in vizinhos:
        fluxo += aresta.fluxo
    return fluxo
```

Caminhos

Ford-Fulkerson



Obtenção de Caminhos

```
def get_caminho(self, inicio: int, fim: int, delta: int = 1) -> List[Aresta]:
    pais = self.bfs(inicio, fim, delta)
    if pais:
        caminho = [pais[-1]]
        atual = pais[fim]
        while atual != inicio:
            caminho.append(atual)
            atual = pais[atual.vertice2]
        return caminho
    else:
        return False
```

Verificação de Elegíveis

```
def elegivel(self, delta: int) -> bool:
    if self.capacidade_residual >= delta:
        return True
    else:
        return False
```

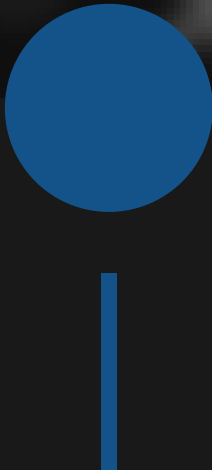
BFS

- VERIFICA ELEGIBILIDADE DA ARESTA
- SEM NECESSIDADE DE RECRIAR O GRAFO RESIDUAL

```
fila = [inicio]
while fila != []:
    atual = fila.pop()
    vizinhos = []
    if setup == 1:
        vizinhos = self.get_vizinhos(atual)
        setup = 0
    else:
        vizinhos = self.get_vizinhos(atual.vertice2)
    for vizinho in vizinhos:
        if explorados[vizinho.vertice2] == 0 and vizinho.elegivel(delta):
            explorados[vizinho.vertice2] = 1
            pais[vizinho.vertice2] = atual
            if vizinho.vertice2 == fim:
                pais.append(vizinho)
                return pais
            fila.append(vizinho)
```

Gargalos

Ford-Fulkerson



Obtenção do Gargalo Aumentar Fluxos no Caminho

```
def get_gargalo(self, caminho: List[Aresta]) -> Aresta:
    gargalo = caminho[0]
    for aresta in caminho:
        if aresta.capacidade_residual < gargalo.capacidade_residual:
            gargalo = aresta
    return gargalo

def aumentar(self, caminho: List[Aresta]) -> None:
    gargalo = self.get_gargalo(caminho)
    gargalo_valor = gargalo.capacidade_residual
    for aresta in caminho:
        aresta.atualizar(gargalo_valor)
```



Atualizar Arestas

```
def atualizar(self, gargalo:int):
    self.fluxo += gargalo
    if self.reversa == True:
        self.capacidade_residual = self.fluxo
    else:
        self.capacidade_residual = self.capacidade - self.fluxo
    self.reversa_pointer.atualizar(gargalo)
```


Alocação de Fluxo das Arestas

```
def alocacao(self, escrita: str) -> int:
    arquivo = open(escrita, 'w')
    arquivo.writelines('aresta,vertice1,vertice2,fluxo\n')
    grafo = self.grafo
    count=0
    inicio = time.time()
    for vertice in grafo:
        for aresta in grafo[vertice]:
            if aresta.reversa == False:
                count += 1
                arquivo.writelines(f' {count},{aresta.vertice1},{aresta.vertice2},{aresta.fluxo} \n')
    fim = time.time()
    tempo=(fim - inicio)
    arquivo.close()
    return tempo
```

Resultados



Grafo	Tempo Médio(s)	Fluxo Máximo
Grafo_1	0,005	284
Grafo_2	0,009	276820
Grafo_3	0,032	291
Grafo_4	0,086	253278
Grafo_5	0,744	618
Grafo_6	2,070	548517
Grafo_7	4,455	611
Grafo_8	136,119	5382665

