UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira
Teoría dos Grafos

Relatório do Projeto

Parte 2

Nome do Integrante	RA
Raphael Iniesta Reis	10396285
Tomás Fiorelli Barbosa	10395687

Relação de Tipos de Pokémons em Grafos

GitHub: < https://github.com/tomasfiorelli/grafos-pokemon >
YouTube: < https://youtu.be/Bwgqj4l80mc >

1. DEFINIÇÃO DO PROBLEMA

O tema escolhido pelo grupo foi a interação de ataque e defesa dos tipos elementais presentes no jogo *Pokémon*. Em resumo, um ataque de qualquer elemento possui um coeficiente de efetividade (sem efeito - 0, pouco efetivo - 0.5, efetivo - 1 ou super efetivo - 2) contra um *Pokémon* de um certo tipo – como por exemplo, um ataque do elemento água é super efetivo contra um *Pokémon* do tipo fogo e é pouco efetivo contra um *Pokémon* do tipo planta (podemos falar também que o *Pokémon* do tipo planta é resistente à ataques do tipo água). Abaixo está uma representação simples do "triangulo inicial" de elementos, pois consiste nos tipos de Pokémon que podem ser escolhidos no começo dos jogos da franquia.

Representação do triângulo de efetividade dos tipos iniciais de Pokémon

Considerando todos os elementos, são definidos 18 tipos únicos e cada um se relaciona com os outros 17 tipos, totalizando um valor de 306 interações entre os diferentes elementos.

OBS: para este projeto, será desconsiderado dois fatores em função de como será trabalhado o grafo produzido. O primeiro é que será desconsiderado a interação de elementos com eles mesmos, algo que pode acontecer normalmente no jogo (por exemplo, fogo é pouco efetivo contra fogo), já que fomos instruídos a não trabalhar com laços; e o segundo é que *Pokémons* podem (ou não) possuir 2 tipos elementais, ocasionando de gerar mais vértices e arestas (171 combinações em que cada interagiria com um dos 18 possíveis tipos de ataques – totalizando em aproximadamente 3.000 interações). Trabalharemos com 1 tipo para reduzir o escopo do trabalho

O projeto desenvolvido também necessitava gerar algum impacto em relação com os Objetivos de Desenvolvimento Sustentável para a humanidade. Contudo, o tema trabalhado não possui uma influência significativa para ser considerado como algum tipo de melhoria para a sociedade em

qualquer um desses objetivos – aquele que melhor se enquadra é o número 9, “Indústria, Inovação e Infraestrutura”, pois buscamos fornecer um mecanismo de análise para uma determinada empresa (*Nintendo/Game Freak/Pokémon Company*), possibilitando-a de inovar na criação de novos tipos ao realizarem uma análise aprofundada das relações dos tipos já existentes.

2. MODELAGEM DO PROBLEMA

O arquivo “grafo.txt” enviado junto de todos os documentos possui a modelagem para o problema das interações dos tipos de *Pokémon*. Ele foi desenvolvido seguindo todos os critérios requisitados pelo enunciado, em que cada linha representa um certo elemento do grafo. Segue abaixo uma parte do arquivo (devido a sua quantidade de linhas, retiramos a maioria das arestas):

```
6
18
NORMAL
FIRE
WATER
ELETRIC
GRASS
ICE
FIGHTING
POISON
GROUND
FLYING
PSYCHIC
BUG
ROCK
GHOST
DRAGON
DARK
STEEL
FAIRY
306
NORMAL FIRE 1.0
NORMAL WATER 1.0
NORMAL ELETRIC 1.0
NORMAL GRASS 1.0
NORMAL ICE 1.0
NORMAL FIGHTING 1.0
NORMAL POISON 1.0
NORMAL GROUND 1.0
NORMAL FLYING 1.0
NORMAL PSYCHIC 1.0
NORMAL BUG 1.0
NORMAL ROCK 0.5
NORMAL GHOST 0.0
NORMAL DRAGON 1.0
NORMAL DARK 1.0
NORMAL STEEL 0.5
NORMAL FAIRY 1.0
FIRE NORMAL 1.0
FIRE WATER 0.5
...
```

3. DESENVOLVIMENTO DA APLICAÇÃO – PARTE 1

Nessa etapa, será apresentado as funcionalidades do programa desenvolvido, seguindo as opções disponíveis no menu

Menu de edição

```
=== RELAÇÃO DOS TIPOS DE POKÉMON EM GRAFO ===

Selecione uma opção:
[1] Ler dados do arquivo grafo.txt
[2] Gravar dados no arquivo grafo.txt
[3] Inserir vértice
[4] Inserir aresta
[5] Remover vértice
[6] Remover aresta
[7] Mostrar conteúdo do arquivo
[8] Mostrar grafo
[9] Apresentar a conexidade do grafo e o reduzido
[0] Encerrar a aplicação

Opção:
```

Ler dados do arquivo grafo.txt

```
69      def lerArquivo(self, nome_arquivo):
70          with open(nome_arquivo, 'r', encoding="utf-8") as arquivo:
71              self.tipo_grafo = int(arquivo.readline())
72
73              quantidade_vertices = int(arquivo.readline().strip())
74              for _ in range(quantidade_vertices):
75                  self.insereV(arquivo.readline().strip())
76
77              quantidade_arestas = int(arquivo.readline())
78              for _ in range(quantidade_arestas):
79                  origem, destino, peso = arquivo.readline().split()
80                  peso_float = float(peso)
81                  self.insereA(origem, destino, peso_float)
```

```
Opção: 1
Digite o nome do arquivo: grafo.txt
Dados do arquivo grafo.txt lidos com sucesso.

Pressione [ENTER] para voltar
```

Gravar dados no arquivo grafo.txt

```python
    def salvarArquivo(self, nome_arquivo):
        with open(nome_arquivo, 'w', encoding="utf-8") as arquivo:
            # Escreve o tipo de grafo
            arquivo.write("6\n")

            # Escreve a quantidade de vértices
            arquivo.write(f"{len(self.adj)}\n")

            # Escreve os rótulos dos vértices
            for vertice in self.adj.keys():
                arquivo.write(f"{vertice}\n")

            # Escreve a quantidade de arestas
            total_arestas = sum(len(vizinhos) for vizinhos in self.adj.values())
            arquivo.write(f"{total_arestas}\n")

            # Escreve as arestas e pesos
            for vertice, vizinhos in self.adj.items():
                for vizinho, peso in vizinhos.items():
                    arquivo.write(f"{vertice} {vizinho} {peso}\n")
```

```
Opção: 2
Digite o nome do arquivo: grafo.txt
Dados gravados no arquivo grafo.txt com sucesso.

Pressione [ENTER] para voltar
```

Inserir vértice; Remove vértice

```python
    def insereV(self, v):
        if v not in self.adj:
            self.adj[v] = {}
        self.qtde_vertices += 1

    def removeV(self, vertice):
        if vertice in self.adj:
            del self.adj[vertice]
            self.qtde_vertices -= 1
            for v in self.adj:  # Remove todas as arestas que envolvem o vértice
                if vertice in self.adj[v]:
                    del self.adj[v][vertice]
                    self.qtde_arestas -= 1
            print(f"Vértice '{vertice}' removido com sucesso e todas as suas arestas relacionadas.")
        else:
            print(f"O vértice '{vertice}' não existe no grafo.")
```

Inserir aresta; Remove aresta

```python
    def insereA(self, v, w, peso=1.0):
        if v not in self.adj:
            self.adj[v] = {}
        if w not in self.adj:
            self.adj[w] = {}
        self.adj[v][w] = peso
        self.qtde_arestas += 1

    def removeA(self, v, w):
        if v in self.adj and w in self.adj[v]:
            del self.adj[v][w]
        self.qtde_arestas -= 1
```

```
Opção: 3
Digite o nome do vértice: KENZO
Vértice 'KENZO' inserido com sucesso.

Pressione [ENTER] para voltar
```

```
Opção: 8

 === MATRIZ DE ADJACÊNCIA ===
     KEN
KEN    -
```

```
Opção: 5
Digite o nome do vértice a ser removido: KENZO
Vértice 'KENZO' removido com sucesso e todas as suas arestas relacionadas.

Pressione [ENTER] para voltar
```

Mostrar conteúdo do arquivo

```python
    def show(self):
        print("\nGRAFO [6] - grafo orientado com peso na aresta")
        print(f"Quantidade de vértices: {self.qtde_vertices}")
        print(f"Quantidade de arestas: {self.qtde_arestas}\n")
        for v, vizinhos in self.adj.items():
            print(f"{v[:3]}: ", end="")
            for vizinho, peso in vizinhos.items():
                print(f"[{vizinho[:3]}] = {peso}", end=" | ")
            print()
        print("\nFim da impressao do grafo.\n\n")
```

```
Opção: 7

GRAFO [6] - grafo orientado com peso na aresta
Quantidade de vértices: 18
Quantidade de arestas: 306

NOR: [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 0.5 | [FAN] = 0.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
FOG: [NOR] = 1.0 | [ÁGU] = 0.5 | [ELÉ] = 1.0 | [PLA] = 2.0 | [GEL] = 2.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 2.0 | [PED] = 0.5 | [FAN] = 1.0 | [DRA] = 0.5 | [SOM] = 1.0 | [AÇO] = 2.0 | [FAD] = 1.0 |
ÁGU: [NOR] = 1.0 | [FOG] = 2.0 | [ELÉ] = 1.0 | [PLA] = 0.5 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 2.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 2.0 | [FAN] = 1.0 | [DRA] = 0.5 | [SOM] = 1.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
ELÉ: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 2.0 | [PLA] = 0.5 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 0.0 | [VOA] = 2.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 0.5 | [SOM] = 1.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
PLA: [NOR] = 1.0 | [FOG] = 0.5 | [ÁGU] = 2.0 | [ELÉ] = 1.0 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 2.0 | [TER] = 0.5 | [VOA] = 0.5 | [PSÍ] = 1.0 | [INS] = 2.0 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 0.5 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
GEL: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 2.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 2.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 2.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
LUT: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 2.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 0.5 | [PSÍ] = 0.0 | [INS] = 0.5 | [PED] = 1.0 | [FAN] = 0.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
VEN: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 0.5 | [GEL] = 2.0 | [LUT] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 0.5 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
TER: [NOR] = 1.0 | [FOG] = 2.0 | [ÁGU] = 1.0 | [ELÉ] = 2.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 0.5 | [VOA] = 0.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 2.0 | [FAN] = 1.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
VOA: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 0.5 | [PLA] = 2.0 | [GEL] = 1.0 | [LUT] = 2.0 | [VEN] = 1.0 | [TER] = 1.0 | [PSÍ] = 1.0 | [INS] = 0.5 | [PED] = 0.0 | [FAN] = 1.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
PSÍ: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 0.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [INS] = 1.0 | [PED] = 1.0 | [FAN] = 0.0 | [DRA] = 1.0 | [SOM] = 2.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
INS: [NOR] = 1.0 | [FOG] = 0.5 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 0.5 | [GEL] = 1.0 | [LUT] = 0.5 | [VEN] = 2.0 | [TER] = 1.0 | [VOA] = 2.0 | [PSÍ] = 1.0 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
PED: [NOR] = 1.0 | [FOG] = 2.0 | [ÁGU] = 1.0 | [ELÉ] = 0.5 | [PLA] = 2.0 | [GEL] = 1.0 | [LUT] = 0.5 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 2.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [FAN] = 1.0 | [DRA] = 1.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 1.0 |
FAN: [NOR] = 0.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 0.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 2.0 | [INS] = 1.0 | [PED] = 1.0 | [DRA] = 1.0 | [SOM] = 0.5 | [AÇO] = 1.0 | [FAD] = 1.0 |
DRA: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 1.0 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 1.0 | [FAN] = 1.0 | [SOM] = 1.0 | [AÇO] = 0.5 | [FAD] = 2.0 |
SOM: [NOR] = 1.0 | [FOG] = 1.0 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 0.5 | [VEN] = 1.0 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 2.0 | [INS] = 2.0 | [PED] = 1.0 | [FAN] = 2.0 | [DRA] = 0.0 | [AÇO] = 1.0 | [FAD] = 1.0 |
AÇO: [NOR] = 2.0 | [FOG] = 0.5 | [ÁGU] = 0.5 | [ELÉ] = 0.5 | [PLA] = 1.0 | [GEL] = 2.0 | [LUT] = 2.0 | [VEN] = 0.0 | [TER] = 2.0 | [VOA] = 0.5 | [PSÍ] = 0.5 | [INS] = 0.5 | [PED] = 0.5 | [FAN] = 1.0 | [DRA] = 0.0 | [SOM] = 1.0 | [FAD] = 1.0 |
FAD: [NOR] = 1.0 | [FOG] = 0.5 | [ÁGU] = 1.0 | [ELÉ] = 1.0 | [PLA] = 1.0 | [GEL] = 1.0 | [LUT] = 2.0 | [VEN] = 0.5 | [TER] = 1.0 | [VOA] = 1.0 | [PSÍ] = 1.0 | [INS] = 1.0 | [PED] = 1.0 | [FAN] = 1.0 | [DRA] = 2.0 | [SOM] = 2.0 | [AÇO] = 0.5 |

Fim da impressao do grafo.
```

Mostrar grafo

```python
    def showMin(self):
        vertices = self.adj.keys()
        print("\n === MATRIZ DE ADJACÊNCIA ===")
        print(" " * 4, end="")
        for v in vertices:
            print(f"{v[:3]:^6}", end="")
        print()
        for v1 in vertices:
            print(f"{v1[:3]:<4}", end="")
            for v2 in vertices:
                if v2 in self.adj.get(v1, {}):
                    print(f"{self.adj[v1][v2]:^6.1f}", end="")
                else:
                    print(f"{'-':^6}", end="")
            print()
        print("\nFim da impressao da matriz de adjacência.\n\n")
```

```
Opção: 8

 === MATRIZ DE ADJACÊNCIA ===
     NOR   FOG   ÁGU   ELÉ   PLA   GEL   LUT   VEN   TER   VOA   PSÍ   INS   PED   FAN   DRA   SOM   AÇO   FAD
NOR    -    1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   0.5   0.0   1.0   1.0   0.5   1.0
FOG  1.0    -    0.5   1.0   2.0   2.0   1.0   1.0   1.0   1.0   1.0   2.0   0.5   1.0   0.5   1.0   2.0   1.0
ÁGU  1.0   2.0    -    1.0   0.5   1.0   1.0   1.0   2.0   1.0   1.0   1.0   2.0   1.0   0.5   1.0   1.0   1.0
ELÉ  1.0   1.0   2.0    -    0.5   1.0   1.0   1.0   0.0   2.0   1.0   1.0   1.0   1.0   0.5   1.0   1.0   1.0
PLA  1.0   0.5   2.0   1.0    -    1.0   1.0   2.0   0.5   0.5   1.0   2.0   1.0   1.0   0.5   1.0   0.5   1.0
GEL  1.0   1.0   1.0   1.0   2.0    -    1.0   1.0   1.0   2.0   1.0   1.0   1.0   1.0   2.0   1.0   0.5   1.0
LUT  1.0   1.0   1.0   1.0   1.0   2.0    -    1.0   1.0   0.5   0.0   0.5   1.0   0.0   1.0   1.0   1.0   1.0
VEN  1.0   1.0   1.0   1.0   0.5   2.0   1.0    -    1.0   1.0   1.0   0.5   1.0   1.0   1.0   1.0   1.0   1.0
TER  1.0   2.0   1.0   2.0   1.0   1.0   1.0   0.5    -    0.0   1.0   1.0   2.0   1.0   1.0   1.0   0.5   1.0
VOA  1.0   1.0   1.0   0.5   2.0   1.0   2.0   1.0   1.0    -    1.0   0.5   0.0   1.0   1.0   1.0   0.5   1.0
PSÍ  1.0   1.0   1.0   1.0   1.0   1.0   0.0   1.0   1.0   1.0    -    1.0   1.0   0.0   1.0   2.0   0.5   1.0
INS  1.0   0.5   1.0   1.0   0.5   1.0   0.5   2.0   1.0   2.0   1.0    -    1.0   1.0   1.0   1.0   1.0   1.0
PED  1.0   2.0   1.0   0.5   2.0   1.0   0.5   1.0   1.0   2.0   1.0   1.0    -    1.0   1.0   1.0   0.5   1.0
FAN  0.0   1.0   1.0   1.0   1.0   1.0   0.0   1.0   1.0   1.0   2.0   1.0   1.0    -    1.0   0.5   1.0   1.0
DRA  1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0    -    1.0   0.5   2.0
SOM  1.0   1.0   1.0   1.0   1.0   1.0   0.5   1.0   1.0   1.0   2.0   2.0   1.0   2.0   0.0    -    1.0   1.0
AÇO  2.0   0.5   0.5   0.5   1.0   2.0   2.0   0.0   2.0   0.5   0.5   0.5   0.5   1.0   0.0   1.0    -    1.0
FAD  1.0   0.5   1.0   1.0   1.0   1.0   2.0   0.5   1.0   1.0   1.0   1.0   1.0   1.0   2.0   2.0   0.5    -
```

4. DESENVOLVIMENTO DA APLICAÇÃO – PARTE 2

Abaixo estão as imagens obtidas das respectivas funcionalidades desenvolvidas para a segunda parte do projeto da disciplina Teoria dos Grafos

Menu de análises:

```
================ MENU DE ANÁLISES ================

Selecione uma opção:
[1] Triângulos perfeitos
[2] Relações completas
[3] Subgrafos de Efetividade
[0] <- Voltar

Opção:
```

Triângulos Perfeitos

Esta análise tem como objetivo encontrar no grafo completo laços que formam um triângulo perfeito, isto é, 3 vértices em que possuem peso igual a 2.0 em um sentido, e 0.5 no sentido inverso, formando um triângulo, assim como na imagem apresentada inicialmente para ilustração, entre FOGO, GRAMA e ÁGUA.

```python
# Parte 2
def triangulos(self):
    # relações de um triângulo perfeito
    # TIPO_1 TIPO_2: 2.0
    # TIPO_2 TIPO_3: 2.0
    # TIPO_3 TIPO_1: 2.0
    # TIPO_2 TIPO_1: 0.5
    # TIPO_3 TIPO_2: 0.5
    # TIPO_1 TIPO_3: 0.5

    triangulos = []
    vertices = list(self.matriz.keys())
    for i in range(len(vertices)):
        for j in range(len(vertices)):
            for k in range(len(vertices)):
                v1 = vertices[i]
                v2 = vertices[j]
                v3 = vertices[k]

                if v1 == v2 or v1 == v3 or v2 == v3:
                    continue

                # Verifica relações
                if (self.matriz[v1][v2] == 2.0 and
                    self.matriz[v2][v3] == 2.0 and
                    self.matriz[v3][v1] == 2.0 and
                    self.matriz[v2][v1] == 0.5 and
                    self.matriz[v3][v2] == 0.5 and
                    self.matriz[v1][v3] == 0.5):
                    # adiciona e ordena caso seja adicionado um
                    # trio já existente e que esteja deslocado
                    triangulos.append(tuple(sorted((v1, v2, v3))))

    # remove trios repetidos
    triangulos = set(triangulos)

    print(f"\n{' Triângulos ':=^25}")
    if triangulos:
        for triangulo in triangulos:
            print(f"{triangulo[0]} >> {triangulo[1]} >> {triangulo[2]}")
    else:
        print("Nenhum triângulo encontrado.")
    print(f"{' ':=^25}")
```

```
Opção: 1

====== Triângulos =======
FIRE >> GRASS >> WATER
FIGHTING >> FLYING >> ROCK
FIRE >> ROCK >> STEEL
GRASS >> GROUND >> POISON
=========== ===========

Pressione [ENTER] para voltar
```

Pudemos encontrar os seguintes resultados: existem 4 triângulos perfeitos dentre os tipos padrões do jogo, sendo eles

FOGO >> PLANTA >> ÁGUA (demonstração)
LUTADOR >> VOADOR >> PEDRA
FOGO >> PEDRA >> AÇO
PLANTA >> TERRESTRE >> VENENOSO

Relações Completas

A segunda análise desenvolvida foi uma ferramenta que percorre todas as relações de um vértice (tanto na vertical, quanto na horizontal, se pensarmos como uma tabela/matriz de valores), categorizando as relações desse vértice em função do peso que foi atribuído nas seguintes classes: “forte contra”, “fraco contra”, “não afeta”, “resistente a”, “vulnerável a” e “imune a”. Abaixo pode-se analisar o código desenvolvido, assim como o resultado de 2 testes, para o tipo PLANTA (GRASS) e o tipo TERRESTRE (GROUND):

```python
def relacoes(self, vertice: str):
    if vertice not in self.matriz:
        print(f'O vértice "{vertice}" não existe no grafo.')
        return

    forte_contra = []
    fraco_contra = []
    nao_afeta = []

    resistente = []
    vulneravel = []
    imune = []

    for destino in self.matriz[vertice].keys():
        if self.matriz[vertice][destino] == 2.0:
            forte_contra.append(destino)
        elif self.matriz[vertice][destino] == 0.5:
            fraco_contra.append(destino)
        elif self.matriz[vertice][destino] == 0.0:
            nao_afeta.append(destino)

    for origem in self.matriz.keys():
        if origem == vertice:
            continue

        if self.matriz[origem][vertice] == 0.5:
            resistente.append(origem)
        elif self.matriz[origem][vertice] == 2.0:
            vulneravel.append(origem)
        elif self.matriz[origem][vertice] == 0.0:
            imune.append(origem)

    print(f"\n{' Relações de ' + vertice + ' ':=^45}")
    print(f"{vertice} -> atacando")
    print(f"- Forte contra:\t {', '.join(forte_contra) or '---'}")
    print(f"- Fraco contra:\t {', '.join(fraco_contra) or '---'}")
    print(f"- Não afeta:\t {', '.join(nao_afeta) or '---'}")
    print(f"\n{vertice} -> defendendo")
    print(f"- Resistente a:\t {', '.join(resistente) or '---'}")
    print(f"- Vulnerável a:\t {', '.join(vulneravel) or '---'}")
    print(f"- Imune a:\t {', '.join(imune) or '---'}")
```

```
Opção: 2
Digite o nome do vertice: GRASS

============= Relações de GRASS =============
GRASS -> atacando
- Forte contra:  WATER, GROUND, ROCK
- Fraco contra:  FIRE, POISON, FLYING, BUG, DRAGON, STEEL
- Não afeta:     ---

GRASS -> defendendo
- Resistente a:  WATER, ELETRIC, GROUND
- Vulnerável a:  FIRE, ICE, POISON, FLYING, BUG
- Imune a:       ---

Pressione [ENTER] para voltar
```

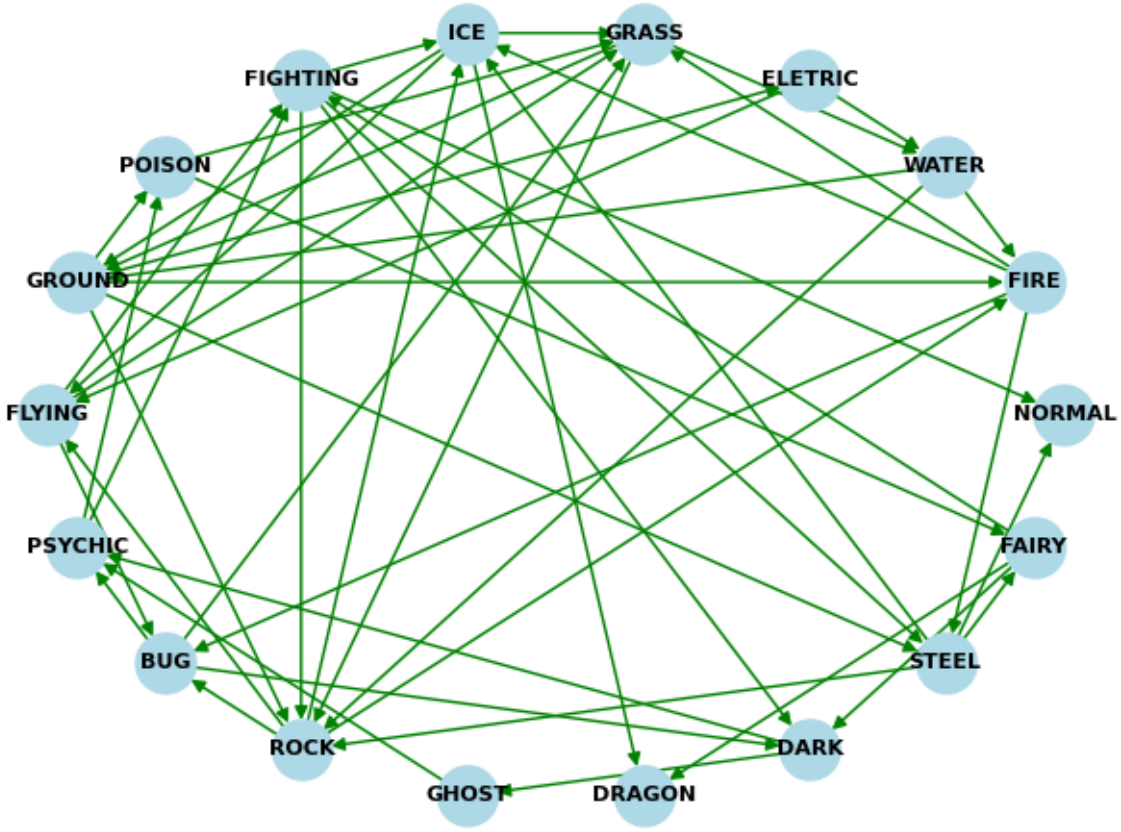
```
Opção: 2
Digite o nome do vertice: GROUND

============ Relações de GROUND =============
GROUND -> atacando
- Forte contra:  FIRE, ELETRIC, POISON, ROCK, STEEL
- Fraco contra:  GRASS, BUG
- Não afeta:     FLYING

GROUND -> defendendo
- Resistente a:  POISON, ROCK
- Vulnerável a:  WATER, GRASS, ICE
- Imune a:       ELETRIC

Pressione [ENTER] para voltar
```

Grafos de Efetividade

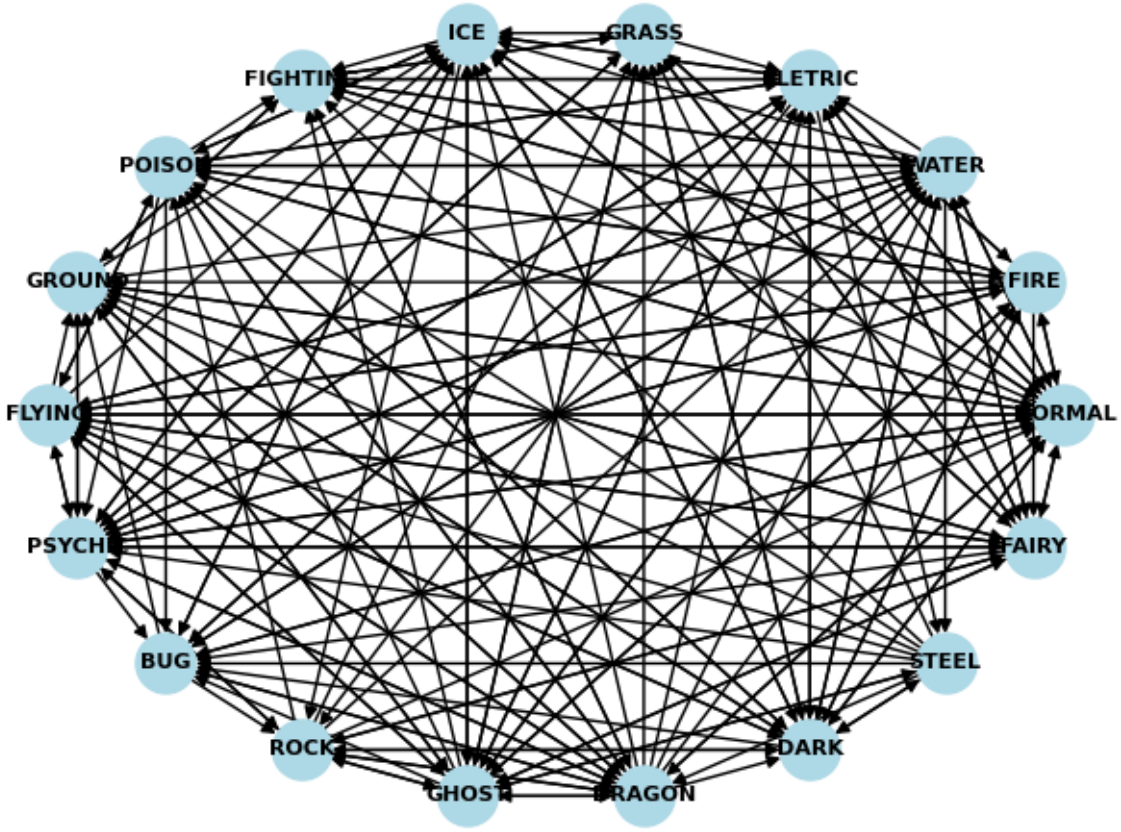
Por fim, foi desenvolvido uma função para geração do grafo em imagens utilizando as bibliotecas NetworkX e Matplotlib. Em função do peso das arestas, está função é capaz de gerar um grafo separado para cada tipo de efetividade de ataque. Apesar desse recurso, conforme o número de vértices e arestas aumenta, existe a possibilidade de gerar uma imagem com conteúdo exagerado, impossibilitando seu entendimento. Abaixo estão os 4 tipos de efetividades em grafos separados, juntamente do código da função:

```python
def gerar_imagem_grafo(self, nome_arquivo: str,
                       super_efetivo=True,
                       efetivo=True,
                       nao_efetivo=True,
                       sem_efeito=True):

    grafo = nx.DiGraph()

    for vertice in self.matriz.keys():
        grafo.add_node(vertice)

    # atribuicao de cores
    for origem, vizinhos in self.matriz.items():
        for destino, peso in vizinhos.items():
            if peso == 2.0 and super_efetivo:
                cor = 'green'
                grafo.add_edge(origem, destino, color=cor)
            elif peso == 1.0 and efetivo:
                cor = 'black'
                grafo.add_edge(origem, destino, color=cor)
            elif peso == 0.5 and nao_efetivo:
                cor = 'red'
                grafo.add_edge(origem, destino, color=cor)
            elif peso == 0.0 and sem_efeito:
                cor = 'purple'
                grafo.add_edge(origem, destino, color=cor)

    pos = nx.circular_layout(grafo)

    nx.draw(grafo, pos, with_labels=True, node_color='lightblue', font_size=8,
            font_weight='bold', node_size=500, edge_color=[data['color'] for (u, v, data) in grafo.edges(data=True)],
            node_shape='o')

    plt.title("Grafo de Tipos de Pokémon")
    plt.tight_layout()

    plt.savefig(nome_arquivo)
    plt.close()
    print(f'A imagem do grafo foi salva em {nome_arquivo}.')
```

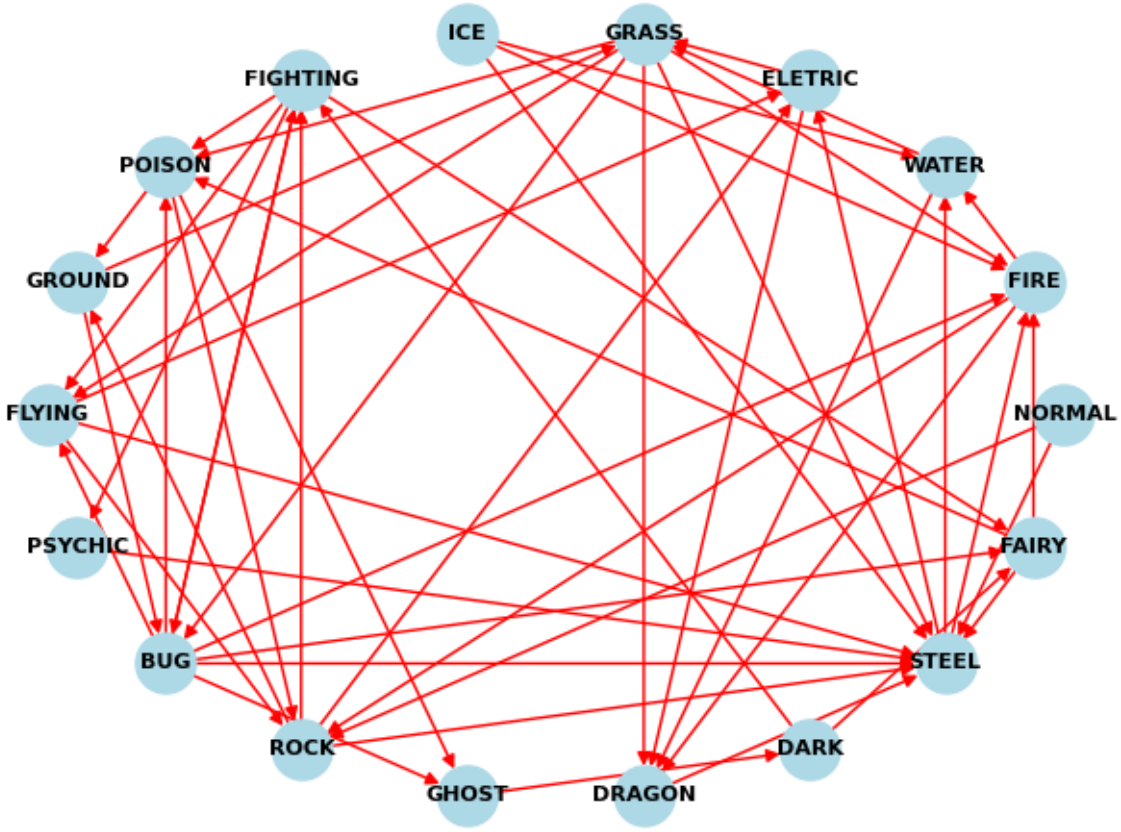
Grafo super efetivo (peso = 2.0)



Grafo efetivo (peso = 1.0)



Grafo não efetivo (relação = 0.5)



Grafo sem efeito (relação = 0.0)