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

Relatório do Projeto

Parte 1

Nome do Integrante	RA
Lucas Meres	**10395777**
Renan Tagliaferro	**10395211**
Thiago Leandro Liporace	**10395816**

Relatório

Análise da Rede Social e Dinâmicas de Amizades Online no Instagram

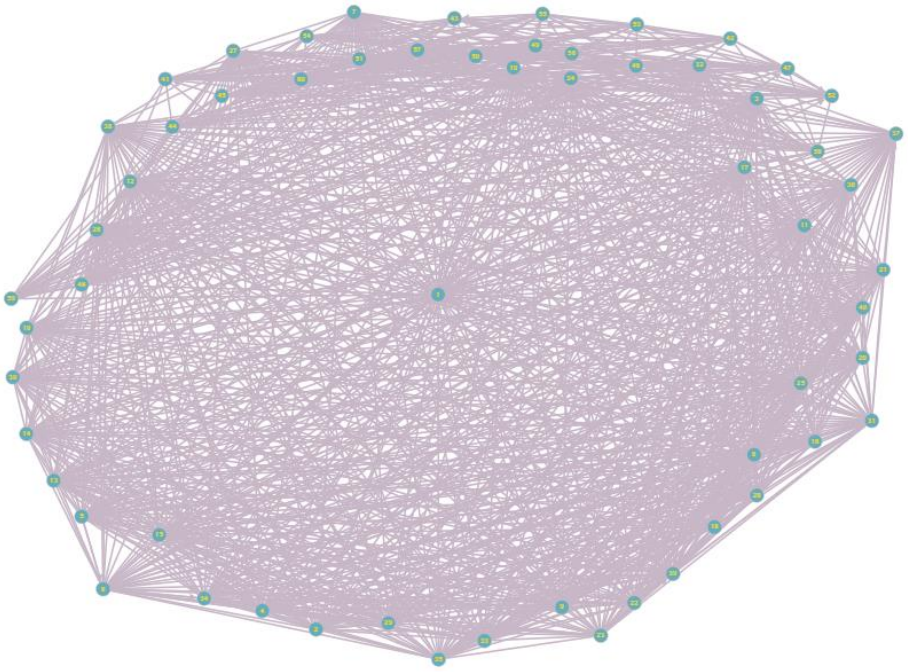
DEFINIÇÃO DO PROBLEMA:

A análise da rede social e das dinâmicas de amizades online no Instagram apresenta uma oportunidade única para explorar as complexidades das interações humanas em um ambiente digital. Esta plataforma, que conta com mais de um bilhão de usuários ativos mensalmente, serve como um vasto campo de estudo para padrões de comportamento, conexões sociais e influência de conteúdo. Cada conta ativa no Instagram pode ser vista como um nó dentro de um extenso grafo, onde as relações de seguir entre os usuários formam as arestas que conectam este intricado tecido social.

O objetivo principal deste estudo é mapear as redes de amizades e analisar as interações de 60 usuários, divididos em 2 grupos sociais diferentes, e analisamos: curtidas, comentários e compartilhamentos. Ao aprofundar na estrutura e dinâmica das redes de amizade online, pretendemos não só mapear as relações existentes, mas também oferecer análises de interações dentro de grupos.

PROJETO:

O Grafo foi modelado no Graph online a partir de uma matriz de adjacência gerada pelo nosso código, o grafo gerado foi o seguinte:
http://graphonline.ru/pt/?graph=rJjFBFioCkTRjpTF



Se trata de um grafo extremamente denso, pelo grande número de interações que ocorrem em uma rede social.
A seguir, descreveremos os passos para chegarmos neste grafo:

1º Passo: o webScraping:
Desenvolvemos um código em python, utilizando a biblioteca instaloader, para automatizar a retirada de dados do aplicativo. O código inteiro se encontra em nosso github, na sua versão sem nome de usuários, e senha para podermos logar no aplicativo pelo código. O core do código se encontra na imagem abaixo, e itera sobre os usuários fornecidos em uma lista, que precisam estar abertos ao perfil usado como ponto de partida, e recolha todos os últimos "n" posts dos usuários, sendo "n" também parametrizável, neste caso rastreamos as 200 últimas postagens. Após, o código relaciona o nome dos usuários que realizaram comentários e "deram like" com os nomes que estão na lista de input de usuários. Ao fim, o código joga as informações em um CSV, dando peso 1 para cada like, e peso 2 para cada comentário.
Imagem: "core" do código pyhton.

```python
# Iterar sobre cada usuário
for usuario in usuarios:
    # Obtenha informações sobre o perfil do usuário
    profile = instaloader.Profile.from_username(loader.context, usuario)

    # Obtenha os seguidores do usuário e filtre apenas os que estão na lista original
    seguidores = [follower.username for follower in profile.get_followers() if follower.username in usuarios]

    # Obtenha as últimas postagens do usuário
    postagens_usuario = []
    for post in profile.get_posts():
        if len(postagens_usuario) >= num_postagens:
            break
        likes_post = [like.username for like in post.get_likes() if like.username in usuarios]
        comentarios_post = [comment.user.username for comment in post.get_comments() if comment.user.username in usuarios]
        postagens_usuario.append({"likes": likes_post, "comentarios": comentarios_post})

    # Adicione interações com outros usuários
    for postagem in postagens_usuario:
        for like in postagem["likes"]:
            interacoes_usuarios[usuario][like] += 1
        for comentario in postagem["comentarios"]:
            interacoes_usuarios[usuario][comentario] += 2

# Salvar o output em um arquivo CSV
with open('output_interacoes.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile, delimiter=';')
    writer.writerow(['usuario'] + usuarios)

    for usuario, interacoes in interacoes_usuarios.items():
        valores_interacoes = [str(interacoes[usuario_int]) for usuario_int in usuarios]
        writer.writerow([usuario] + valores_interacoes)

print("Output salvo em 'output_interacoes.csv'.")
```

O arquivo de output então, por linha, contém o id do usuário, seguido da soma dos likes e comentários dos outros usuários na ordem que eles se encontram nas linhas do CSV.

Por exemplo:
Usuário_x 0 1 2
Usuário_y 10 1
Usuário_z 2 2 0

Neste exemplo, o usuário_x obteve (entre comentários e likes) peso 0 de si mesmo (excluímos comentários nos próprios posts), peso 1 do Usuário_y e 2 do usuário_z, e assim por diante.
Por fim, utilizamos o notepad++, para trocar cada ocorrência de um nome de usuário no arquivo, por um número que os identificaria, que no caso também corresponderia a seu índice na matrix + 1, e obtivemos um arquivo como o demonstrado na imagem abaixo:

Imagem: como ficaram as primeiras 15 linhas do arquivo resultante:

```
1;0;18;17;20;13;19;9;20;23;21;5;0;20;22;16;9;22;22;13;10;4;1;5;3;9;4;1;16;5;8;20;14;15;26;24;1;7;6;11;4;16;16;4;10;0;20;8;22;5;21;22;25;11;0;1;10;20;18;26;15
2;15;0;8;18;4;7;15;21;4;11;5;24;0;11;1;10;11;10;7;23;4;2;18;3;2;7;4;12;1;23;1;5;11;9;23;4;6;1;15;10;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
3;1;9;0;23;1;20;24;1;21;23;8;17;1;26;8;4;2;15;24;3;5;9;4;10;10;3;23;3;9;13;18;8;14;1;4;15;10;17;5;4;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
4;13;14;21;0;3;2;2;23;4;7;21;24;12;19;1;12;2;20;24;6;18;10;13;26;13;3;19;17;14;25;23;18;1;26;2;4;1;20;1;22;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
5;0;11;19;2;0;3;9;3;22;3;0;1;21;26;3;10;12;7;18;18;21;17;14;1;18;13;19;11;4;13;1;26;2;12;7;20;5;16;23;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
6;19;12;17;3;11;0;20;2;12;0;20;4;10;10;18;1;17;21;9;18;14;11;19;13;14;21;20;23;15;25;20;15;7;5;5;10;6;25;12;18;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
7;15;21;22;14;21;3;0;15;0;13;14;18;0;25;0;2;12;10;11;25;25;9;18;13;5;12;7;4;18;21;16;26;25;0;10;13;21;13;17;21;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
8;15;4;13;4;3;2;22;0;4;12;6;2;10;5;9;12;26;22;19;4;2;2;20;17;1;9;1;14;3;14;20;25;19;25;0;23;17;2;19;21;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
9;14;14;13;24;8;22;25;8;0;6;17;12;25;8;21;15;9;4;16;12;7;20;6;21;1;20;21;24;26;23;5;9;10;9;22;23;17;7;21;25;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
10;2;0;26;16;9;21;5;7;14;0;10;20;21;19;15;16;20;24;26;18;23;23;12;5;22;21;17;19;12;24;13;26;15;3;15;5;12;9;26;19;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
11;23;9;1;6;2;16;11;22;13;11;0;2;25;23;15;4;18;25;10;10;26;23;13;15;11;5;3;16;6;12;15;25;24;14;16;19;5;5;20;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
12;8;20;19;6;16;23;26;23;21;25;23;0;10;21;25;25;6;3;1;11;9;2;16;23;15;3;12;23;8;7;16;24;15;25;16;10;14;1;26;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
13;12;24;23;22;8;21;20;14;24;10;25;22;0;1;3;18;16;6;20;12;3;0;18;17;4;16;22;3;20;24;18;20;9;5;5;4;13;26;13;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
14;12;23;14;7;24;18;14;2;13;7;3;6;23;0;21;23;0;0;18;4;9;15;11;18;24;16;23;17;2;11;3;18;23;15;22;19;1;13;22;3;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
15;15;19;7;21;15;17;6;5;6;25;9;15;13;9;0;6;0;26;18;6;1;2;26;9;14;14;4;6;16;6;18;8;21;10;15;5;14;22;11;19;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
```

Note que como o usuário 1 foi o ponto de partida, ele possui relações com quase todos os usuários. E por pegarmos 40 pessoas de um grupo e 20 de outro, nas demais linhas, muito raramente ocorrem relações entre os dois grupos.

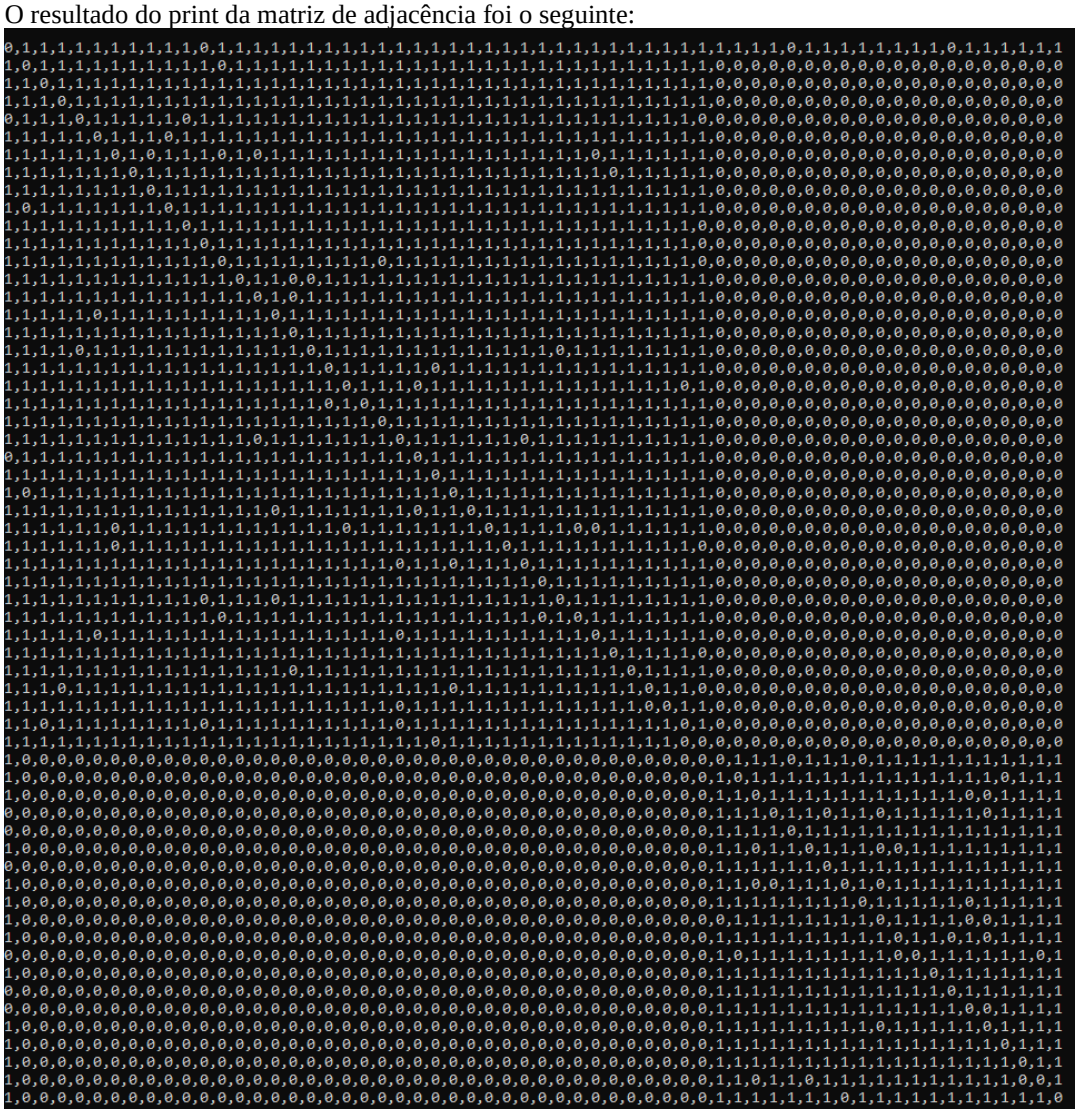
O próximo passo foi transformar este arquivo, em um arquivo na formatação igual ao grafo.txt pedido no enunciado do projeto.
Fizemos um código em C++ que transforma um arquivo como o descrito acima no formato desejado, o código também se encontra no github, mas por motivos de brevidade não será incluso screenshots deste código. Após transformarmos o txt no formato de entrada desejado, utilizamos o método desenvolvido para o projeto chamado FileToGraph(), que transforma o grafo.txt em uma matriz de adjacência. O código está na imagem abaixo:
Imagem: código que transforma o grafo.txt em matriz de adjacência.

```
TGrafo& TGrafo::FileToGraph(std::string fileName)
{
    std::ifstream file(fileName);

    if (!file.is_open())
    {
        std::cerr << "Erro ao abrir o arquivo." << std::endl;
        TGrafo grafo(0);
        return grafo;
    }
    int ignore;
    file >> ignore;//pula primeira linha.
    int a, v = 0;
    file >> v;// le a segunda linha == vertices
    for (int i = 0; i < v; i++)
        file >> ignore;//ja que o id e o numero da linnha são iguais o id é irrelevante de ser guardado

    file >> a;//le o num de arestas

    TGrafo* grafo = new TGrafo(v);

    // Lê as arestas do arquivo e atualiza a matriz, m é o num de linhas a serem lidas
    this->m = 0;
    for (int i = 0; i < a; ++i)
    {
        int x, y, z = 0;
        file >> x >> y >> z;
        grafo->insereA(x-1, y-1, z);//-1 pq o indice começa no zero, mas a linha no 1.
    }
    file.close();
    return *grafo;
}
```

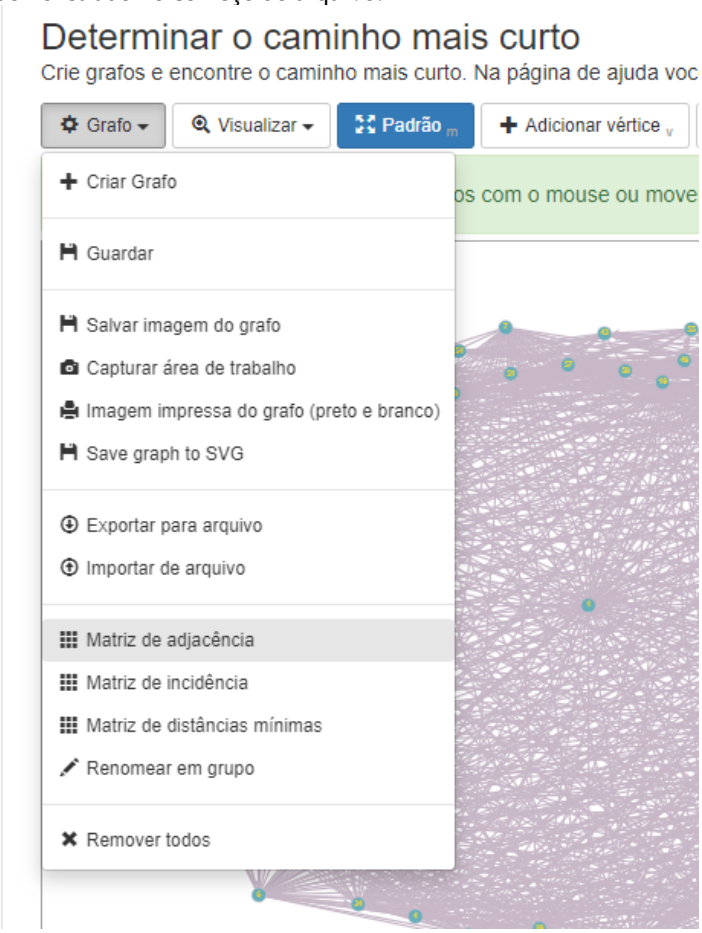
Após este passo, com a matriz de adjacência formada, basta imprimi-la no formato aceito pelo site GraphOnline, onde independente do peso, atribuímos 1 para onde há aresta e zero para onde não existam arestas. Foi desenvolvido o código abaixo:

```
void TGrafo::ShowMatrixOnly()
{
    for (int i = 0; i < n; i++)
    {
        std::cout << "\n";
        for (int w = 0; w < n; w++)
            if (w == n-1)
            {
                if (adj[i][w] == INT_MAX)
                    std::cout << "0";

                else std::cout << "1";
            }
            else
            {
                if (adj[i][w] == INT_MAX)

                    std::cout <<"0" << ",";

                else std::cout  << "1" << ",";
            }
    }
    std::cout << "\nfim da impressao do grafo." << std::endl;
}
```

O resultado do print da matriz de adjacência foi o seguinte:



Depois apenas utilizamos a função “matriz de adjacência” do graphonline, colamos a matriz acima, e ele gerou o grafo demonstrado no começo do arquivo.



E Assim geramos nosso grafo estudado.

Objetivos da ODS contemplados:

A análise da rede social Instagram, embora pareça distante dos objetivos tradicionalmente associados a desafios globais como pobreza, saúde e educação, a influência das redes sociais no comportamento humano e nas decisões coletivas pode ser significativa, como detalhado abaixo.

ODS 3: Saúde e Bem-estar
Objetivo: Garantir uma vida saudável e promover o bem-estar para todos, em todas as idades.
Justificativa: A análise do comportamento dos usuários no Instagram pode oferecer insights sobre o impacto das redes sociais na saúde mental e emocional das pessoas. Ao identificar padrões de uso que contribuem para o estresse, a ansiedade e a depressão, o projeto pode propor recomendações para promover um ambiente digital mais saudável, alinhando-se com o objetivo de melhorar o bem-estar geral.

ODS 9: Indústria, Inovação e Infraestrutura
Objetivo: Construir infraestruturas resilientes, promover a industrialização inclusiva e sustentável e fomentar a inovação.
Justificativa: O desenvolvimento de ferramentas analíticas avançadas para estudar o Instagram envolve inovação tecnológica e pode promover melhorias na infraestrutura digital. Este objetivo é atendido pelo projeto ao incentivar o uso de tecnologias inovadoras para análise de dados, contribuindo para a criação de uma indústria digital mais sustentável e inclusiva.

TESTES EXECUÇÃO DO MENU
Os testes estão no arquivo Testes.cpp, basta descomentar a linha da main que chama a função “ExecutarTestes”, que os testes serão executados.
Faremos os testes com outros grafos de resultados já conhecidos, pelo grafo do projeto ser muito grande e denso.

Ao executar o programa, o seguinte menu se abre:

```
**********************************************************
*** PROJETO GRAFOS ***
*** Analise de interacoes no Instagram ***
Atencao, o programa ja gerou o grafo original a partir do arquivo txt ao iniciar!(ou seja opcao 1 ja rodada)
Digite o numero da opcao
[1] Ler dados do arquivo grafo.txt e Criar Matriz de Adjacencia
[2] Gravar Matriz de Adjacencia no arquivo grafo.txt
[3] Inserir vertice
[4] Inserir aresta
[5] Remove vertice
[6] Remove aresta
[7] Mostrar conteudo do arquivo
[8] Mostrar grafo
[9] Apresentar a conexidade do grafo e o grafo reduzido
[0] Encerrar a aplicacao
Option:
```

Por padrão o arquivo com a matriz de adjacência original é carregado lago que o programa é executado, ou seja a opção 1 já é executada.
Segue abaixo o código responsável pela função 1

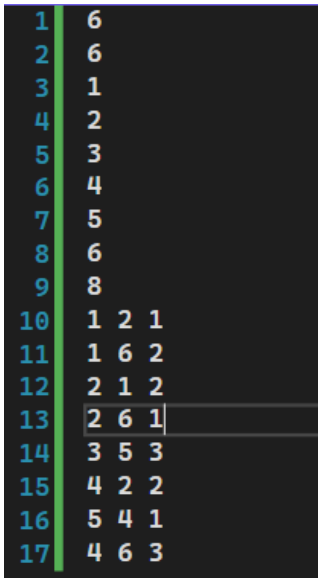
```
TGrafo& TGrafo::FileToGraph(std::string fileName)
{
    std::ifstream file(fileName);

    if (!file.is_open())
    {
        std::cerr << "Erro ao abrir o arquivo." << std::endl;
        TGrafo grafo(0);
        return grafo;
    }
    int ignore;
    file >> ignore;//pula primeira linha.
    int a, v = 0;
    file >> v;// le a segunda linha == vertices
    for (int i = 0; i < v; i++)
        file >> ignore;//ja que o id e o numero da linnha são iguais o id é irrelevante de ser guardado

    file >> a;//le o num de arestas

    TGrafo* grafo = new TGrafo(v);

    // Lê as arestas do arquivo e atualiza a matriz, m é o num de linhas a serem lidas
    this->m = 0;
    for (int i = 0; i < a; ++i)
    {
        int x, y, z = 0;
        file >> x >> y >> z;
        grafo->insereA(x-1, y-1, z);//-1 pq o indice começa no zero, mas a linha no 1.
    }
    file.close();
    return *grafo;
}
```

Já demonstramos que ele funciona anteriormente, porém, faremos 2 testes com arquivos novos:
Testes opção 1:
Foi executado desta maneira:

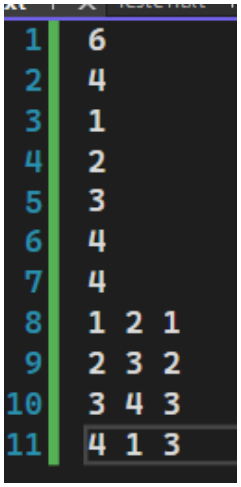
```
std::cout << "Teste ex 1:";
std::cout << "Lendo o arquivo e construindo matriz ... \n";
TGrafo grafo = grafo.FileToGraph("./Teste1.txt");
std::cout << "\nmatriz 1 obtida a partir de arquivo\n";
grafo.show();
std::cout << "_____________________________________________________________________________\n";
std::cout << "Lendo o arquivo e construindo matriz ... \n";
TGrafo grafo2 = grafo2.FileToGraph("./Teste2.txt");
std::cout << "\nmatriz 2 obtida a partir de arquivo\n";
grafo2.show();
```

Com estes arquivos txt:

Teste 1:



Teste 2:



Resultado:

```
Teste ex 1:Lendo o arquivo e construindo matriz...

matriz 1 obtida a partir de arquivo
n: 6
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf
fim da impressao do grafo.
--------------------------------------------------------------------------------
Lendo o arquivo e construindo matriz...

matriz 2 obtida a partir de arquivo
n: 4
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf
fim da impressao do grafo.
```

Testes Opção 2
Testes executados desta maneira:

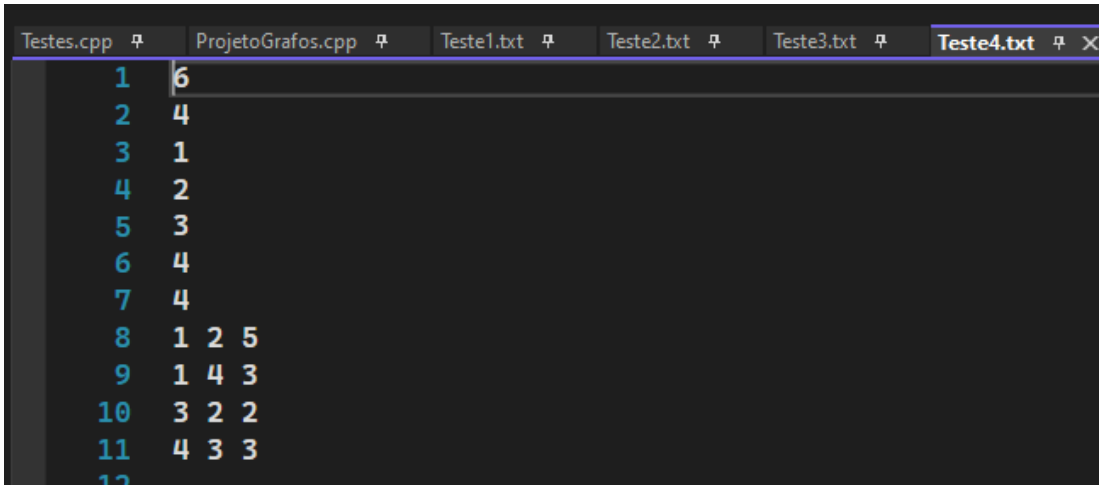
```
TGrafo grafo3(3);
grafo3.insereA(0, 1, 1);
grafo3.insereA(0, 2, 2);
grafo3.insereA(1, 2, 5);
grafo3.insereA(2, 1, 3);

TGrafo grafo4(4);
grafo4.insereA(0, 1, 5);
grafo4.insereA(0, 3, 3);
grafo4.insereA(2, 1, 2);
grafo4.insereA(3, 2, 3);
grafo4.insereA(3, 4, 1);

grafo3.MatrixToFile("./Teste3.txt");
grafo4.MatrixToFile("./Teste4.txt");
```

Resultado dos arquivos txt criados:
Teste 3.txt

Testes.cpp | ProjetoGrafos.cpp | Teste1.txt | Teste2.txt | **Teste3.txt**

```
1  6
2  3
3  1
4  2
5  3
6  4
7  1 2 1
8  1 3 2
9  2 3 5
10 3 2 3
11
```

Teste4.txt



Teste Opção 3:
Código responsável pela opção 3:

```
void TGrafo::InsertVertix()
{
    int v = this->n;
    std::cout << "\nNumero atual de vertices: " << v << ".\n";
    std::cout << "\nindice do novo vertice: " << v << ". Id do novo vertice: \n" << v + 1 << ".\n";
    int newSize = v + 1;
    float** oldMatrix = this->adj;
    float** newMatrix = new float* [newSize];

    for (int i = 0; i < newSize; ++i)
    {
        newMatrix[i] = new float[newSize];
        for (int j = 0; j < newSize; ++j)
        {
            if (i < v && j < v)
                // Copiar a matriz antiga
                newMatrix[i][j] = oldMatrix[i][j];

            else
                // Inicializar novas células com INT_MAX
                newMatrix[i][j] = INT_MAX;
        }
    }

    // Deletar a matriz antiga
    for (int i = 0; i < v; ++i)
        delete[] oldMatrix[i];
    delete[] oldMatrix;

    this->adj = newMatrix;
    this->n = newSize;
}
```

Foram utilizados os grafos criados pelo teste da opção 1, executados desta maneira:

```
std::cout << "\nGrafo 1 original: \n";
grafo.show();
grafo.InsertVertix();
std::cout << "\nGrafo 1 apos insercao: \n";
grafo.show();
std::cout << "\nGrafo 2 original: \n";
grafo2.show();
grafo2.InsertVertix();
std::cout << "\nGrafo 2 apos insercao: \n";
grafo2.show();
```

Resultado da execução:

```
Grafo 1 original:
n: 6
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf
fim da impressao do grafo.

Numero atual de vertices: 6.

indice do novo vertice: 6. Id do novo vertice:
7.

Grafo 1 apos insercao:
n: 7
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 2 original:
n: 4
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf
fim da impressao do grafo.

Numero atual de vertices: 4.

indice do novo vertice: 4. Id do novo vertice:
5.

Grafo 2 apos insercao:
n: 5
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= inf
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.
```

Teste opção 4:
Com base nos grafo aumentado pelo teste da opção 3, vamos inserir arestas da seguinte maneira:

```
  std::cout << "\nGrafo 1 atual: \n";
  grafo.show();
  grafo.insereA(1,6,3);
  std::cout << "\nGrafo 1 apos insercao: \n";
  grafo.show();

  std::cout << "\nGrafo 2 atual: \n";
  grafo2.show();
  grafo2.insereA(2, 4, 5);
  std::cout << "\nGrafo 2 apos insercao: \n";
  grafo2.show();
```

Resultado:

```
Grafo 1 atual:
n: 7
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 1 apos insercao:
n: 7
m: 9

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= 3
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 2 atual:
n: 5
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= inf
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.

Grafo 2 apos insercao:
n: 5
m: 5

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= 5
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.
```

Teste Opcao 5:
Foi executado desta maneira, com base nos grafos do teste da opção 2:

```
  std::cout << "\nGrafo 3 atual: \n";
  grafo3.show();
  grafo3.RemoveV(0);
  std::cout << "\nGrafo 3 apos insercao: \n";
  grafo3.show();

  std::cout << "\nGrafo 4 atual: \n";
  grafo4.show();
  grafo4.RemoveV(1);
  std::cout << "\nGrafo 4 apos insercao: \n";
  grafo4.show();
```

Este foi o resultado:

```
Grafo 3 atual:
n: 3
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= 2
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 5
Adj[2,0]= inf Adj[2,1]= 3 Adj[2,2]= inf
fim da impressao do grafo.

Grafo 3 apos insercao:
n: 2
m: 4

Adj[0,0]= inf Adj[0,1]= 5
Adj[1,0]= 3 Adj[1,1]= inf
fim da impressao do grafo.

Grafo 4 atual:
n: 4
m: 4

Adj[0,0]= inf Adj[0,1]= 5 Adj[0,2]= inf Adj[0,3]= 3
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf
Adj[2,0]= inf Adj[2,1]= 2 Adj[2,2]= inf Adj[2,3]= inf
Adj[3,0]= inf Adj[3,1]= inf Adj[3,2]= 3 Adj[3,3]= inf
fim da impressao do grafo.

Grafo 4 apos insercao:
n: 3
m: 4

Adj[0,0]= inf Adj[0,1]= inf Adj[0,2]= 3
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= inf
Adj[2,0]= inf Adj[2,1]= 3 Adj[2,2]= inf
fim da impressao do grafo.
```

Teste Opção 6:
Feito em cima dos grafos 1 e 2.

```
    std::cout << "\nGrafo 1 atual: \n";
    grafo.show();
    grafo.removeA(1,6);
    std::cout << "\nGrafo 1 apos insercao: \n";
    grafo.show();

    std::cout << "\nGrafo 2 atual: \n";
    grafo2.show();
    grafo2.removeA(0,1);
    std::cout << "\nGrafo 2 apos insercao: \n";
    grafo2.show();
```

Resultados:

```
Grafo 1 atual:
n: 7
m: 9

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= 3
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 1 apos insercao:
n: 7
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 2 atual:
n: 5
m: 5

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= 5
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.

Grafo 2 apos insercao:
n: 5
m: 4

Adj[0,0]= inf Adj[0,1]= inf Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= 5
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.
```

Teste Opção 7
Mostraremos o conteúdo dos 4 arquivos de teste: executado desta maneira:

```
  std::cout << "\nprint dos 4 arquivos:\n";
  std::cout << "\narquivo 1:\n";
  PrintTxt("./teste1.txt");
  std::cout << "\narquivo 2:\n";
  PrintTxt("./teste2.txt");
  std::cout << "\narquivo 3:\n";
  PrintTxt("./teste3.txt");
  std::cout << "\narquivo 4:\n";
  PrintTxt("./teste4.txt");
```

Resultados:

```
print dos 4 arquivos:

arquivo 1:

Tipo do Grafo:
6 - grafo orientado com peso na aresta

numero de vertices: 6
ids dos vertices:
1, 2, 3, 4, 5, 6.

numero de Arestas: 8
conexoes das Arestas: (formato origem->destino|peso)
1->2|Peso: 1
1->6|Peso: 2
2->1|Peso: 2
2->6|Peso: 1
3->5|Peso: 3
4->2|Peso: 2
5->4|Peso: 1
4->6|Peso: 3

arquivo 2:

Tipo do Grafo:
6 - grafo orientado com peso na aresta

numero de vertices: 4
ids dos vertices:
1, 2, 3, 4.

numero de Arestas: 4
conexoes das Arestas: (formato origem->destino|peso)
1->2|Peso: 1
2->3|Peso: 2
3->4|Peso: 3
4->1|Peso: 3
```

```
arquivo 3:

Tipo do Grafo:
6 - grafo orientado com peso na aresta

numero de vertices: 3
ids dos vertices:
1, 2, 3.

numero de Arestas: 4
conexoes das Arestas: (formato origem->destino|peso)
1->2|Peso: 1
1->3|Peso: 2
2->3|Peso: 5
3->2|Peso: 3

arquivo 4:

Tipo do Grafo:
6 - grafo orientado com peso na aresta

numero de vertices: 4
ids dos vertices:
1, 2, 3, 4.

numero de Arestas: 4
conexoes das Arestas: (formato origem->destino|peso)
1->2|Peso: 5
1->4|Peso: 3
3->2|Peso: 2
4->3|Peso: 3
```

Teste Opção 8:
Foi executado desta maneira:

```
std::cout << "\nMostrando Grafos:\n";
std::cout << "\nGrafo 1:\n";
grafo.show();
std::cout << "\nGrafo 2:\n";
grafo2.show();
```

Resultados:

```
Mostrando Grafos:

Grafo 1:
n: 7
m: 8

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf Adj[0,5]= 2 Adj[0,6]= inf
Adj[1,0]= 2 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf Adj[1,4]= inf Adj[1,5]= 1 Adj[1,6]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= inf Adj[2,4]= 3 Adj[2,5]= inf Adj[2,6]= inf
Adj[3,0]= inf Adj[3,1]= 2 Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf Adj[3,5]= 3 Adj[3,6]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= 1 Adj[4,4]= inf Adj[4,5]= inf Adj[4,6]= inf
Adj[5,0]= inf Adj[5,1]= inf Adj[5,2]= inf Adj[5,3]= inf Adj[5,4]= inf Adj[5,5]= inf Adj[5,6]= inf
Adj[6,0]= inf Adj[6,1]= inf Adj[6,2]= inf Adj[6,3]= inf Adj[6,4]= inf Adj[6,5]= inf Adj[6,6]= inf
fim da impressao do grafo.

Grafo 2:
n: 5
m: 4

Adj[0,0]= inf Adj[0,1]= inf Adj[0,2]= inf Adj[0,3]= inf Adj[0,4]= inf
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= 2 Adj[1,3]= inf Adj[1,4]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 3 Adj[2,4]= 5
Adj[3,0]= 3 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf Adj[3,4]= inf
Adj[4,0]= inf Adj[4,1]= inf Adj[4,2]= inf Adj[4,3]= inf Adj[4,4]= inf
fim da impressao do grafo.
```

Teste Opção 9:
Executado desta maneira:

```
TGrafo grafoC1(4);
grafoC1.insereA(1, 0, 1);
grafoC1.insereA(2, 0, 1);
grafoC1.insereA(2, 3, 1);
grafoC1.insereA(2, 1, 1);
int c1Result = grafoC1.graphCategory();
std::cout << "\nGrafo imputado\n";
grafoC1.show();
std::cout << "\n(c1)Este Grafo eh do tipo : " << Exaux(c1Result);
TGrafo grafoReduzido = grafoC1.GetReducedMatrix();
grafoReduzido.show();
std::cout << "_____\n";

TGrafo grafoC3(4);
grafoC3.insereA(0, 2, 1);
grafoC3.insereA(1, 0, 1);
grafoC3.insereA(2, 1, 1);
grafoC3.insereA(2, 3, 1);
grafoC3.insereA(3, 0, 1);
int c3Result = grafoC3.graphCategory();
std::cout << "\nGrafo imputado\n";
grafoC3.show();
std::cout << "\n(c3)Este Grafo eh do tipo : " << Exaux(c3Result);
TGrafo grafoReduzido2 = grafoC3.GetReducedMatrix();
grafoReduzido2.show();
```

Resultado:

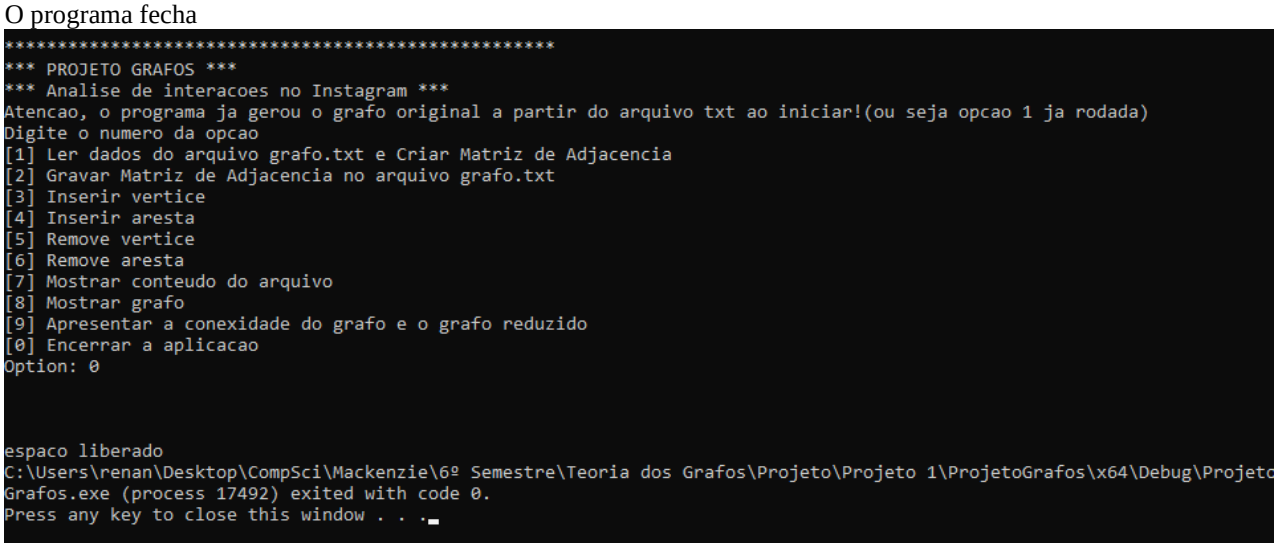
```
Grafo imputado
n: 4
m: 4

Adj[0,0]= inf Adj[0,1]= inf Adj[0,2]= inf Adj[0,3]= inf
Adj[1,0]= 1 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf
Adj[2,0]= 1 Adj[2,1]= 1 Adj[2,2]= inf Adj[2,3]= 1
Adj[3,0]= inf Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf
fim da impressao do grafo.

(c1)Este Grafo eh do tipo : C1!
Componente Fortemente Conectado encontrado, indice: 0 : Membros: 3
Componente Fortemente Conectado encontrado, indice: 1 : Membros: 4
Componente Fortemente Conectado encontrado, indice: 2 : Membros: 2
Componente Fortemente Conectado encontrado, indice: 3 : Membros: 1
n: 4
m: 4

Adj[0,0]= inf Adj[0,1]= 1 Adj[0,2]= 1 Adj[0,3]= 1
Adj[1,0]= inf Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf
Adj[2,0]= inf Adj[2,1]= inf Adj[2,2]= inf Adj[2,3]= 1
Adj[3,0]= inf Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf
fim da impressao do grafo.
_____

Grafo imputado
n: 4
m: 5

Adj[0,0]= inf Adj[0,1]= inf Adj[0,2]= 1 Adj[0,3]= inf
Adj[1,0]= 1 Adj[1,1]= inf Adj[1,2]= inf Adj[1,3]= inf
Adj[2,0]= inf Adj[2,1]= 1 Adj[2,2]= inf Adj[2,3]= 1
Adj[3,0]= 1 Adj[3,1]= inf Adj[3,2]= inf Adj[3,3]= inf
fim da impressao do grafo.

(c3)Este Grafo eh do tipo : C3!
Componente Fortemente Conectado encontrado, indice: 0 : Membros: 1 2 3 4
n: 1
m: 0

Adj[0,0]= inf
fim da impressao do grafo.
```

Agora executando com o nosso grafo encontrado ao analisar instagram:

O interessante é que como todos os contatos partem do primeiro, grafo se torna C3, cujo grafo reduzido é o próprio grafo.

Teste Opção 0:

O programa fecha



----------PARTE 2 DO PROJETO----------

Levando em consideração o enunciado da parte 2, decidimos implementar 5 das técnicas aprendidas nas aulas teóricas ao nosso grafo, sendo elas:
1 - Mostrar os graus de todos os vértices.
2 - Mostrar coloração e partição de vértices (WelshPowell).
3- Verificar se o Grafo é permite ciclo, e se permitir, qual é ele.
4 - Algoritmo de Djikstra aplicado ao grafo.
5 - Algoritmo de fluxo máximo de FordFulkerson.

Desta maneira, o menu inicial foi aumentado, acomodando estas novas possibilidades:
Código das novas possibilidades na main:

```
case 10:
    std::cout << "\nGRAUS DE TODOS OS VERTICES: \n\n";
    grafo.GetAllDegrees();
    break;
case 11:
    std::cout << "\nCOLORACAO COMPLETA DO GRAFO: ";
    grafo.WelshPowell();
    break;
case 12:
    std::cout << "Ciclo Euleriano :\n";
    if (grafo.EulerianPossible())
    {
        std::cout << "O grafo admite Ciclo euleriano.\n";
        std::cout << "Ciclo Euleriano: \n";
        grafo.EulerianCycle(0);
        grafo.FileToGraph("./grafo.txt");
    }
    else
        std::cout << "O grafo nao admite ciclo euleriano.\n";
    break;
case 13:
    std::cout << "\n DJIKSTRA APLICADO AO GRAFO, A PARTIR DO VERTICE 20";
    std::cout << "\nTabela de menores caminhos a partir do vertice 20 do Grafo: \n";
    grafo.dijkstra(19);
    break;
    break;
case 14:
    std::cout << "\n FORD-FULKERSON UTILIZANDO 1 COMO ORIGEM E 60 COMO SUMIDOURO";
    int maxflux = grafo.FordFulkerson(0,59);
    std::cout << "\n\n Fluxo maximo encontrado = " << maxflux<<"\n\n";
    break;
```

Novas opções no menu do console:

```
**************************************************
*** PROJETO GRAFOS ***
*** Analise de interacoes no Instagram ***
Atencao, o programa ja gerou o grafo original a partir do arquivo txt ao iniciar!(ou seja opcao 1 ja rodada)
Digite o numero da opcao
[1] Ler dados do arquivo grafo.txt e Criar Matriz de Adjacencia
[2] Gravar Matriz de Adjacencia no arquivo grafo.txt
[3] Inserir vertice
[4] Inserir aresta
[5] Remove vertice
[6] Remove aresta
[7] Mostrar conteudo do arquivo
[8] Mostrar grafo
[9] Apresentar a conexidade do grafo e o grafo reduzido
[10] PARTE 2 PROJETO: Grau dos vertices:
[11] PARTE 2 PROJETO: Coloracao :
[12] PARTE 2 PROJETO: Percurso/Grafo Euleriano :
[13] PARTE 2 PROJETO: DJIKSTRA APLICADO AO GRAFO. :
[14] PARTE 2 PROJETO: FORD-FULKERSON. :
[0] Encerrar a aplicacao
Option:
```

Opção 10 : mostrar todos os Graus dos vértices:
Foi implementado um código simples, quem itera sobre todos os vértices, e utiliza 2 metódos previamente implementados, o de achar grau interno e externo de um vértice:

```
void TGrafo::GetAllDegrees()
{
    for (int i = 0; i < n; i++)
    {
        std::cout << "Vertice: " << i <<"\n";
        std::cout << "Grau Entrada: " << inDegree(i) << "\n";
        std::cout << "Grau Saida:" << outDegree(i) << "\n\n";
    }
}
```

inDegree e outDegree: já demonstrados anteriormente:

```cpp
int TGrafo::inDegree(int v)
{
    if (v > n - 1 || v < 0)//checa se terá out-of-index
    {
        std::cout << "\nO vertice de indice (" << v << ") que foi passado de entrada nao existe: ";
        return -1;
    }

    int degree = 0; //o grau
    for (int i = 0; i < n; i++)//percorre cada vertice para ver se aresta vinda do v chega nele
        if (adj[i][v] != INT_MAX) // se for != INTMAX tem V chegando nele
            degree++; //incremente o grau
    return degree;
}

int TGrafo::outDegree(int v)
{
    if (v > n - 1 || v < 0)//checa se terá out-of-index
    {
        std::cout << "\nO vertice de indice (" << v << ") que foi passado de entrada nao existe: ";
        return -1;
    }

    int degree = 0; //o grau
    for (int i = 0; i < this->n; i++)//percorre as arestas na linha de vertice indicado
        if (adj[v][i] != INT_MAX) // se for != INTMAX tem alguem chegando nele
            degree++; //incremente o grau
    return degree;
}
```

Execução (parcial):

```
GRAUS DE TODOS OS VERTICES:

Vertice: 0
Grau Entrada: 51
Grau Saida:56

Vertice: 1
Grau Entrada: 37
Grau Saida:38

Vertice: 2
Grau Entrada: 38
Grau Saida:39

Vertice: 3
Grau Entrada: 38
Grau Saida:39

Vertice: 4
Grau Entrada: 38
Grau Saida:36

Vertice: 5
Grau Entrada: 37
Grau Saida:38

Vertice: 6
Grau Entrada: 37
Grau Saida:35

Vertice: 7
Grau Entrada: 39
Grau Saida:38

Vertice: 8
Grau Entrada: 38
Grau Saida:39

Vertice: 9
Grau Entrada: 38
Grau Saida:38

Vertice: 10
Grau Entrada: 38
Grau Saida:38

Vertice: 11
Grau Entrada: 36
Grau Saida:38

Vertice: 12
Grau Entrada: 36
Grau Saida:37
```

E continua printando até o 60...

Pela execução, conseguimos concluir que o grau dos vértices foi relativamente alto, por se tratar de uma rede social onde fora analizados muitas interações, de pessoas que interagem bastante na internet.

Opção 11: Coloração completa do Grafo:
Utilizamos o algoritmo de WelshPowell para obter a coloração, e o código abaixo já se encontra com comentários explicativos:

```cpp
void TGrafo::WelshPowell()
{
    std::vector<int> cores(n, -1); // Inicializa vetor de cores com -1 (indicando não colorido)
    std::vector<bool> naoColoridos(n, true); // Inicializa vetor de booleanos para indicar vértices não coloridos

    int k = 0; // Número atual da classe de cor

    // Enquanto houver vértices não coloridos
    while (std::find(naoColoridos.begin(), naoColoridos.end(), true) != naoColoridos.end())
    {
        std::vector<int> Ck; // Vetor de vértices da classe de cor k

        // Itera sobre os vértices não coloridos
        for (int i = 0; i < n; i++)
        {
            if (naoColoridos[i])
            {
                bool podeColorir = true;

                // Verifica se algum vizinho de i já foi colorido com a cor k
                for (int j = 0; j < n; j++)
                {
                    if (adj[i][j] != INT_MAX && cores[j] == k)
                    {
                        podeColorir = false;
                        break;
                    }
                }

                // Se nenhum vizinho foi colorido com a cor k, adiciona i à classe de cor k
                if (podeColorir)
                {
                    Ck.push_back(i);
                    cores[i] = k; // Marca i com a cor k
                    naoColoridos[i] = false; // Marca i como colorido
                }
            }
        }

        // Imprime os vértices da classe de cor k
        std::cout << "Cor " << k + 1 << " vertices {";
        for (size_t i = 0; i < Ck.size(); i++)
        {
            std::cout << Ck[i] + 1;
            if (i != Ck.size() - 1)
                std::cout << ",";
        }
        std::cout << "}" << std::endl;
        k++; // Avança para a próxima classe de cor
    }

    // Exibe as cores atribuídas aos vértices
    std::cout << "\nNumero de cores diferentes: " << k << "\n\n";
    for (int i = 0; i < n; i++)
        std::cout << "Vertice " << i + 1 << " tem a cor numero " << cores[i] + 1 << std::endl;

}
```

Após a Execução, temos o número de cores e qual vértice pertence a cada cor. Por motivos práticos, números foram associados as cores ao invés de nomes:

```
COLORACAO COMPLETA DO GRAFO: Cor 1 vertices {1,5,44}
Cor 2 vertices {2,10,41,50}
Cor 3 vertices {3,39,42,52}
Cor 4 vertices {4,37,43,46,59}
Cor 5 vertices {6,16,45}
Cor 6 vertices {7,28,47}
Cor 7 vertices {8,48,60}
Cor 8 vertices {9,49}
Cor 9 vertices {11,51}
Cor 10 vertices {12,32,53}
Cor 11 vertices {13,33,54}
Cor 12 vertices {14,55}
Cor 13 vertices {15,23,56}
Cor 14 vertices {17,36,57}
Cor 15 vertices {18,58}
Cor 16 vertices {19,21}
Cor 17 vertices {20}
Cor 18 vertices {22}
Cor 19 vertices {24,27}
Cor 20 vertices {25,40}
Cor 21 vertices {26,30}
Cor 22 vertices {29}
Cor 23 vertices {31}
Cor 24 vertices {34}
Cor 25 vertices {35}
Cor 26 vertices {38}

Numero de cores diferentes: 26
```

Parte 2 do print:

```
Vertice 1 tem a cor numero 1
Vertice 2 tem a cor numero 2
Vertice 3 tem a cor numero 3
Vertice 4 tem a cor numero 4
Vertice 5 tem a cor numero 1
Vertice 6 tem a cor numero 5
Vertice 7 tem a cor numero 6
Vertice 8 tem a cor numero 7
Vertice 9 tem a cor numero 8
Vertice 10 tem a cor numero 2
Vertice 11 tem a cor numero 9
Vertice 12 tem a cor numero 10
Vertice 13 tem a cor numero 11
Vertice 14 tem a cor numero 12
Vertice 15 tem a cor numero 13
Vertice 16 tem a cor numero 5
Vertice 17 tem a cor numero 14
Vertice 18 tem a cor numero 15
Vertice 19 tem a cor numero 16
Vertice 20 tem a cor numero 17
Vertice 21 tem a cor numero 16
Vertice 22 tem a cor numero 18
Vertice 23 tem a cor numero 13
Vertice 24 tem a cor numero 19
Vertice 25 tem a cor numero 20
Vertice 26 tem a cor numero 21
Vertice 27 tem a cor numero 19
Vertice 28 tem a cor numero 6
Vertice 29 tem a cor numero 22
Vertice 30 tem a cor numero 21
Vertice 31 tem a cor numero 23
Vertice 32 tem a cor numero 10
Vertice 33 tem a cor numero 11
Vertice 34 tem a cor numero 24
Vertice 35 tem a cor numero 25
Vertice 36 tem a cor numero 14
Vertice 37 tem a cor numero 4
Vertice 38 tem a cor numero 26
Vertice 39 tem a cor numero 3
Vertice 40 tem a cor numero 20
Vertice 41 tem a cor numero 2
Vertice 42 tem a cor numero 3
Vertice 43 tem a cor numero 4
Vertice 44 tem a cor numero 1
Vertice 45 tem a cor numero 5
Vertice 46 tem a cor numero 4
Vertice 47 tem a cor numero 6
Vertice 48 tem a cor numero 7
Vertice 49 tem a cor numero 8
Vertice 50 tem a cor numero 2
Vertice 51 tem a cor numero 9
Vertice 52 tem a cor numero 3
Vertice 53 tem a cor numero 10
Vertice 54 tem a cor numero 11
Vertice 55 tem a cor numero 12
Vertice 56 tem a cor numero 13
Vertice 57 tem a cor numero 14
Vertice 58 tem a cor numero 15
Vertice 59 tem a cor numero 4
Vertice 60 tem a cor numero 7
```

Após a Execução podemos concluir que pela alta quantidade de vértices, precisamos de muitas cores para separar o grafo por coloração completamente, novamente, por se tratar de redes sociais ativas.

Opção 12: Ciclo Euleriano.
Decidimos tomar uma abordagem que leva em consideração as regras necessárias para poder haver um ciclo euleriano, antes de ver qual é este ciclo Euleriano no grafo:

Em qualquer grafo existe sempre um número par de vértices de grau impar, E Um grafo conexo ou categoria C3 apresenta um caminho Euleriano(aberto oufechado) se e somente se não ocorrer nenhum ou existir no máximo dois vértices com grau ímpar, E a soma dos graus dos vértices em um grafo é igual a duas vezes o número de arestas.
Isto está refletido no código abaixo:

```
bool TGrafo::EulerianPossible()
{
    int grauImpar = 0;
    int somaGrau = 0;
    int con = this->isConnected();
    if (con == 1)
    {
        std::cout << "Grafo desconexo\n";
        return false;
    }
    for (int i = 0; i < n; ++i)
    {
        int degree = 0;
        for (int j = 0; j < n; ++j)
        {
            if (adj[i][j] != INT_MAX)
                degree++;
        }
        somaGrau += degree;
        if (degree % 2 != 0)
            grauImpar++;
    }
    //Em qualquer grafo existe sempre um número par de vértices de grau impar &&
    //Um grafo conexo ou categoria C3 apresenta um caminho Euleriano(aberto ou
    //fechado) se e somente se não ocorrer nenhum ou existir no máximo dois vértices com grau ímpar.
    if (grauImpar > 2 || grauImpar == 1)
    {
        std::cout << "Grafo com numero de vertices de grau impar ("<<grauImpar <<") \n";
        return false;
    }

    if (somaGrau != n * 2)//a soma dos graus dos vértices em um grafo é igual a duas vezes o número de arestas
    {
        std::cout << "Grafo com a soma dos graus dos vertices diferente de 2 vezes o numero de arestas"
            " soma dos Graus: " << somaGrau << "n arestas:" << m << " \n";
        return false;
    }
    return true;
}
```

Se, e somente Se o grafo permitir ciclo euleriano, usamos a estratégia de remover os vértices por qual passamos, pois eles não podem se repetir, como demonstra a chamada de código recursivo abaixo, que também já imprime o caminho:

```
void TGrafo::EulerianCycleStart()
{
    float** originalGraph = new float* [n];//copiando a matrix
    for (int i = 0; i < n; ++i)
    {
        originalGraph[i] = new float[n];
        for (int j = 0; j < n; ++j)
            originalGraph[i][j] = adj[i][j];
    }

    int originalM = this->m, totalEdges, start, end;//setando variaveis utilizadas para checagem de ciclo válido
    bool foundECycle = false;

    for (int i = 0; i < this->n; i++)//para cada vertice, ele tenta achar um ciclo válido
    {
        start = i;//marca o inicio
        totalEdges = 0;//marca quantos vetices foram percorridos
        this->EulerianCycle(0,totalEdges,end);//tenta achar ciclo para a variável atual
        if (totalEdges == originalM && start == end)//se TODOS os verices foram percorridos E se o inicia for igual ao fim(ciclo)
        {
            std::cout << "\nCAMINHO ACEITO, ESTE CAMINHO EH UM CICLO EULERIANO VALIDO\n";
            foundECycle = true;
            break;//sai do for pois já achou um ciclo
        }
    }

    if(!foundECycle)//printa caso não foram achados ciclos válidos
        std::cout << "NAO FOI POSSIVEL ENCONTRAR UM CAMINHO EULERIANO VALIDO";

    this->adj = originalGraph;//restaurando a matriz
    this->m = originalM;
}
```

A recursão propriamente dita:

```
void TGrafo::EulerianCycle(int v,int &totalEdges, int& last)
{
    for (int i = 0; i < n; ++i)//a apartir do v inicial
    {
        if (adj[v][i] != INT_MAX)
        {
            std::cout << v << " -> " << i << std::endl;
            removeA(v, i);//remove a aresta existente
            totalEdges++;//conta mais uma nas removidas
            EulerianCycle(i, totalEdges, last);//recursivamente tenta chegar até o fim.
        }
    }
    last = v;//ultimo ponto encontrado na recursão.
}
```

Obviamente, pelo código acima “destruir” o grafo, nós carregamos ele de novo a partir do TXT na main:

```
case 12:
    std::cout << "Ciclo Euleriano :\n";
    if (grafo.EulerianPossible())
    {
        std::cout << "O grafo admite Ciclo euleriano.\n";
        std::cout << "Ciclo Euleriano: \n";
        grafo.EulerianCycleStart();
        grafo.FileToGraph("./grafo.txt");
    }
    else
        std::cout << "O grafo nao eh euleriano.\n";
    break;
```

Execução:

```
Option: 12

Ciclo Euleriano :
Grafo com numero de vertices de grau impar, impar(33)
O grafo nao eh euleriano.
```

Como esperado pelo grupo, por conter arestas demais, o grafo não possui o ciclo Euleriano possível, como demonstrado acima por quebrar a regra de número de vértices com grau impar ser ímpar.

Opção 13: Djikstra

Apesar dos pesos do nosso grafo não fazerem sentido na execução de caminho mínimo, decidimos implementar Djikstra para demonstrar que entendemos o funcionamento do algoritmo, e que nosso código é escalável, pois funciona para uma matriz de 3600 elementos:

```cpp
void TGrafo::dijkstra(int src)
{
    std::vector<int> dist(this->n, INT_MAX); // Distância da origem a i
    std::vector<bool> F(this->n, false); // Se o vértice i está fechado
    std::vector<int> rot(this->n, -1); // Predecessor de i no caminho mais curto

    dist[src] = 0; // Distância do vértice de origem para si mesmo é 0

    for (int count = 0; count < this->n - 1; count++)
    {
        // Encontrar o vértice mais próximo (min_index) do conjunto de vértices ainda não processados
        int min = INT_MAX, min_index = 0;

        for (int v = 0; v < this->n; v++)
            if (F[v] == false && dist[v] <= min)
                min = dist[v], min_index = v;

        int u = min_index;
        F[u] = true; // Marca o vértice como processado
        // Atualizar a distância dos vértices adjacentes do vértice selecionado
        for (int v = 0; v < this->n; v++)
            // Atualizar dist[v] apenas se não estiver em F, houver uma aresta de
            // u a v, e o peso total do caminho de src a v através de u é menor
            // que o valor atual de dist[v]
            if (!F[v] && adj[u][v] && dist[u] != INT_MAX
                && dist[u] + adj[u][v] < dist[v])
            {
                dist[v] = dist[u] + adj[u][v];
                rot[v] = u;
            }
    }
    // Imprimir as distâncias e caminhos
    std::cout << "\nVertice\t|Menor Distancia da Origem\t|Caminho" << std::endl;
    for (int i = 0; i < this->n; i++)
    {
        std::cout << i + 1 << "\t|" << dist[i] << "\t\t\t\t|";
        PrintRota(&rot, i);
        std::cout << "\n";
    }
}
```

Metodo acessório para deixar o print mais estético:

```cpp
void TGrafo::PrintRota(std::vector<int>& rot, int j)
{
    // Base case: se j é a origem
    if (rot[j] == -1)
    {
        std::cout << j + 1;
        return;
    }
    PrintRota(&rot, rot[j]);
    std::cout << " -> " << j + 1;
}
```

Execução a partir do vértice 20:

```
 DJIKSTRA APLICADO AO GRAFO, A PARTIR DO VERTICE 20
Tabela de menores caminhos a partir do vertice 20 do Grafo:

Vertice |Menor Distancia da Origem      |Caminho
1       |4                             |20 -> 34 -> 30 -> 1
2       |6                             |20 -> 34 -> 30 -> 38 -> 2
3       |6                             |20 -> 34 -> 30 -> 3
4       |6                             |20 -> 40 -> 4
5       |6                             |20 -> 32 -> 5
6       |6                             |20 -> 34 -> 8 -> 6
7       |7                             |20 -> 40 -> 7
8       |4                             |20 -> 34 -> 8
9       |4                             |20 -> 9
10      |4                             |20 -> 35 -> 10
11      |6                             |20 -> 40 -> 22 -> 11
12      |3                             |20 -> 12
13      |4                             |20 -> 34 -> 30 -> 13
14      |4                             |20 -> 31 -> 39 -> 14
15      |4                             |20 -> 31 -> 39 -> 15
16      |6                             |20 -> 31 -> 19 -> 37 -> 16
17      |7                             |20 -> 17
18      |6                             |20 -> 12 -> 18
19      |4                             |20 -> 31 -> 19
20      |0                             |20
21      |5                             |20 -> 40 -> 21
22      |4                             |20 -> 40 -> 22
23      |5                             |20 -> 34 -> 30 -> 25 -> 27 ->
24      |6                             |20 -> 40 -> 33 -> 24
25      |3                             |20 -> 34 -> 30 -> 25
26      |4                             |20 -> 31 -> 39 -> 26
27      |4                             |20 -> 34 -> 30 -> 25 -> 27
28      |4                             |20 -> 34 -> 30 -> 38 -> 28
29      |6                             |20 -> 31 -> 39 -> 14 -> 29
30      |2                             |20 -> 34 -> 30
```

Resto da impressão:

```
31      |2                              |20 -> 31
32      |3                              |20 -> 32
33      |5                              |20 -> 40 -> 33
34      |1                              |20 -> 34
35      |3                              |20 -> 35
36      |3                              |20 -> 31 -> 36
37      |5                              |20 -> 31 -> 19 -> 37
38      |3                              |20 -> 34 -> 30 -> 38
39      |3                              |20 -> 31 -> 39
40      |3                              |20 -> 40
41      |6                              |20 -> 34 -> 30 -> 1 -> 55 -> 41
42      |10                             |20 -> 34 -> 30 -> 1 -> 55 -> 42
43      |8                              |20 -> 34 -> 30 -> 1 -> 43
44      |10                             |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 44
45      |9                              |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 57 -> 45
46      |6                              |20 -> 34 -> 30 -> 1 -> 55 -> 46
47      |8                              |20 -> 34 -> 30 -> 1 -> 55 -> 41 -> 59 -> 47
48      |8                              |20 -> 34 -> 30 -> 1 -> 55 -> 48
49      |8                              |20 -> 34 -> 30 -> 1 -> 55 -> 49
50      |8                              |20 -> 34 -> 30 -> 1 -> 55 -> 50
51      |8                              |20 -> 34 -> 30 -> 1 -> 55 -> 51
52      |9                              |20 -> 34 -> 30 -> 1 -> 55 -> 51 -> 52
53      |7                              |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 53
54      |10                             |20 -> 34 -> 30 -> 1 -> 55 -> 48 -> 54
55      |5                              |20 -> 34 -> 30 -> 1 -> 55
56      |10                             |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 60 -> 56
57      |7                              |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 57
58      |9                              |20 -> 34 -> 30 -> 1 -> 55 -> 58
59      |7                              |20 -> 34 -> 30 -> 1 -> 55 -> 41 -> 59
60      |7                              |20 -> 34 -> 30 -> 1 -> 55 -> 46 -> 60
```

Apesar de não haver muito sentido em nosso trabalho, é interessante observar pelo resultado que por pertencer a um grupo social diferente das pessoas do índice 40 para cima, o índice 20 precisa necessariamente passar pelo 1 (perfil utilizado para fazer o webscrap) para chegar nos outros, pois em tese estas pessoas nem se conhecem, só tem um conhecido em comum (vértice 1).

Opção 14: Ford-Fulkerson, usando 1 como origem e 60 como sumidouro.
Codigo:

```
int TGrafo::FordFulkerson(int s, int t)
{
    int** rGraph = new int* [n];
    for (int i = 0; i < n; i++)
        rGraph[i] = new int[n];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            rGraph[i][j] = adj[i][j];

    int parent[60];
    int max_flow = 0;
```

```cpp
// Enquanto houver um caminho de aumento do vértice de origem para o vértice de destino
while (bfs(rGraph, s, t, parent))
{
    int path_flow = INT_MAX;
    for (int v = t; v != s; v = parent[v])
    {
        int u = parent[v];
        path_flow = std::min(path_flow, rGraph[u][v]);
    }

    // Atualiza as capacidades residuais do caminho encontrado
    for (int v = t; v != s; v = parent[v])
    {
        int u = parent[v];
        rGraph[u][v] -= path_flow;
        rGraph[v][u] += path_flow;
    }

    // Adiciona o fluxo de caminho ao fluxo máximo
    max_flow += path_flow;

    // Imprime o caminho encontrado
    std::cout << "Caminho encontrado:";
    for (int v = t; v != s; v = parent[v])
    {
        int u = parent[v];
        std::cout << " " << u << "->" << v;
    }
    std::cout << std::endl;

    // Imprime o fluxo máximo atual
    std::cout << "Fluxo maximo atual: " << max_flow << std::endl;
}

    // Libera a memória alocada para a matriz residual
    for (int i = 0; i < n; i++)
        delete[] rGraph[i];
    delete[] rGraph;

    return max_flow;
}
```

Ele utiliza o código auxiliar de busca em profundidade, demonstrado abaixo:

```cpp
bool TGrafo::bfs(int** rGraph, int s, int t, int parent[])
{
    bool visited[60];
    memset(visited, 0, sizeof(visited));

    std::queue<int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v = 0; v < n; v++)
        {
            if (visited[v] == false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    return (visited[t] == true);
}
```

Execução

```
 FORD-FULKERSON UTILIZANDO 1 COMO ORIGEM E 60 COMO SUMIDOUROCaminho encontrado: 1->60
Fluxo maximo atual: 15
Caminho encontrado: 41->60 1->41
Fluxo maximo atual: 23
Caminho encontrado: 42->60 1->42
Fluxo maximo atual: 34
Caminho encontrado: 43->60 1->43
Fluxo maximo atual: 38
Caminho encontrado: 44->60 1->44
Fluxo maximo atual: 39
Caminho encontrado: 46->60 1->46
Fluxo maximo atual: 40
Caminho encontrado: 47->60 1->47
Fluxo maximo atual: 42
Caminho encontrado: 48->60 1->48
Fluxo maximo atual: 44
Caminho encontrado: 49->60 1->49
Fluxo maximo atual: 49
Caminho encontrado: 50->60 1->50
Fluxo maximo atual: 52
Caminho encontrado: 51->60 1->51
Fluxo maximo atual: 63
Caminho encontrado: 52->60 1->52
Fluxo maximo atual: 71
Caminho encontrado: 53->60 1->53
```

Cont:

```
Fluxo maximo atual: 78
Caminho encontrado: 55->60 1->55
Fluxo maximo atual: 79
Caminho encontrado: 56->60 1->56
Fluxo maximo atual: 88
Caminho encontrado: 57->60 1->57
Fluxo maximo atual: 97
Caminho encontrado: 58->60 1->58
Fluxo maximo atual: 107
Caminho encontrado: 59->60 1->59
Fluxo maximo atual: 110
Caminho encontrado: 43->60 41->43 1->41
Fluxo maximo atual: 115
Caminho encontrado: 54->60 41->54 1->41
Fluxo maximo atual: 118
Caminho encontrado: 45->60 42->45 1->42
Fluxo maximo atual: 123
Caminho encontrado: 45->60 44->45 1->44
Fluxo maximo atual: 130
Caminho encontrado: 49->60 44->49 1->44
Fluxo maximo atual: 132
Caminho encontrado: 49->60 46->49 1->46
Fluxo maximo atual: 137
Caminho encontrado: 54->60 46->54 1->46
Fluxo maximo atual: 142
Caminho encontrado: 55->60 46->55 1->46
Fluxo maximo atual: 146
Caminho encontrado: 55->60 47->55 1->47
Fluxo maximo atual: 152


 Fluxo maximo encontrado = 152
```

FIM.

APENDICE:

LINK GITHUB:

https://github.com/RenanTagliaferro/Projeto1Grafos

**FAVOR LEIA O README.

Link Video YOUTUBE: www.youtube.com