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

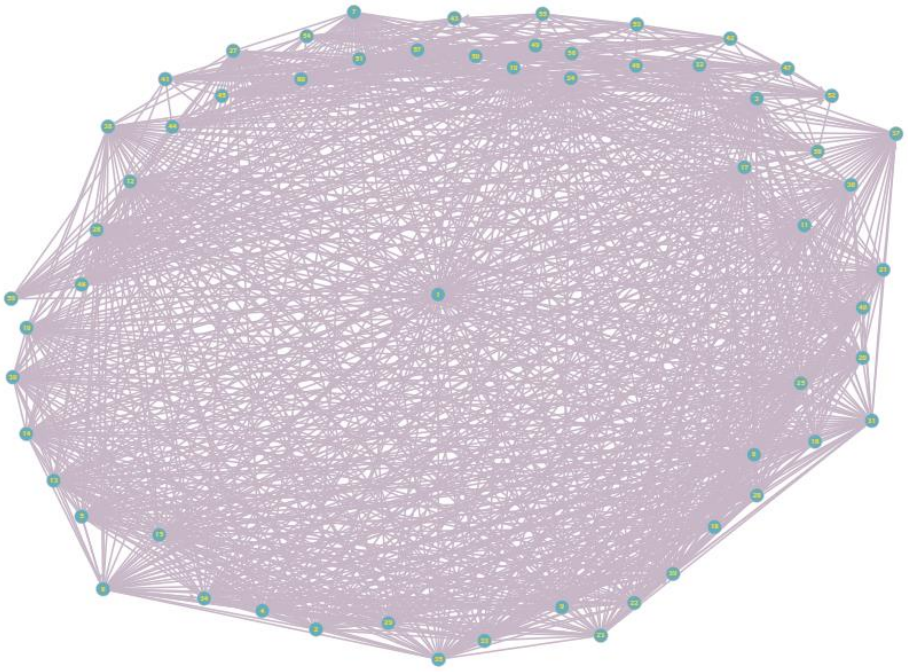
DEFINIÇÃO DO PROBLEMA:

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.

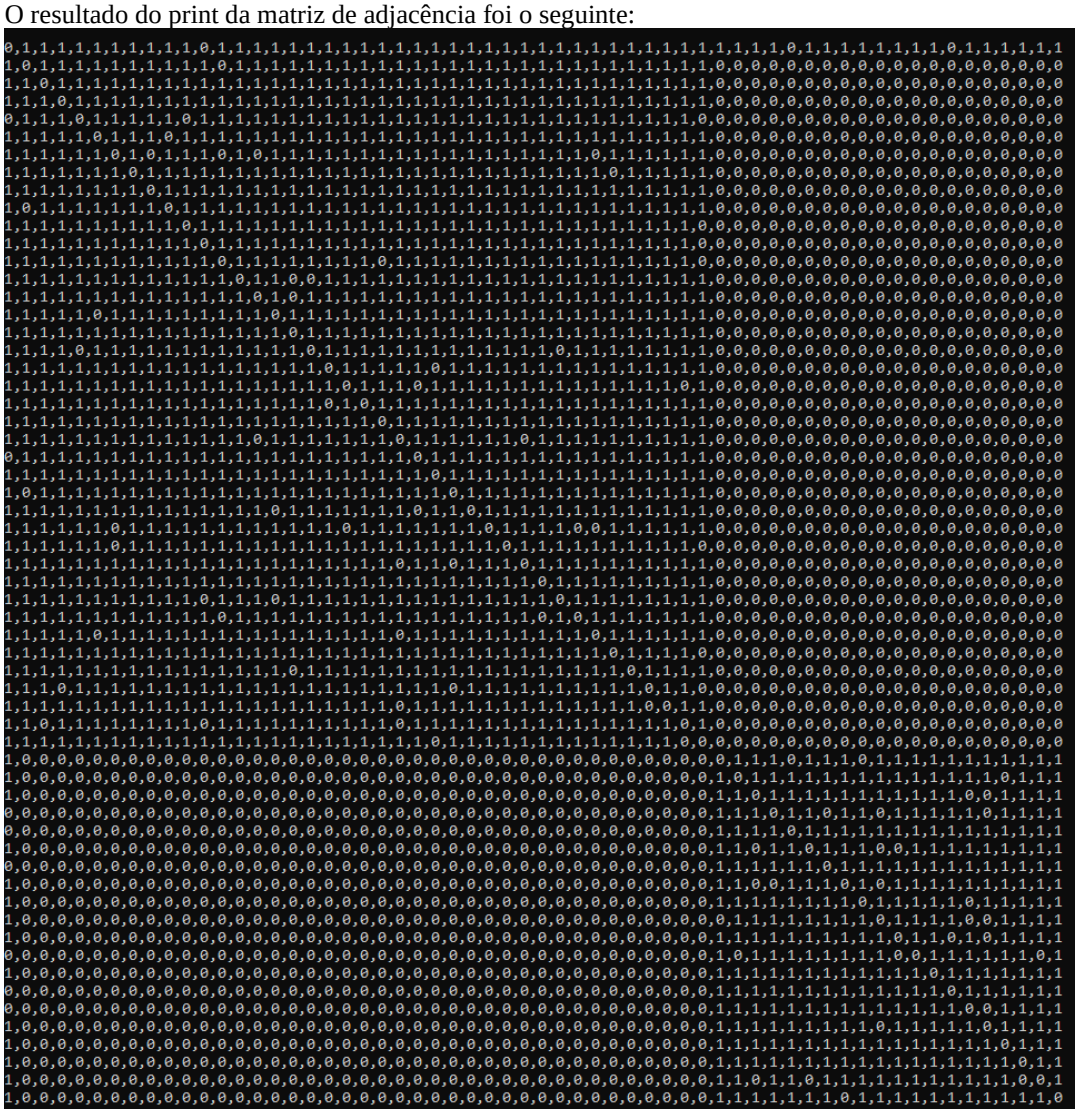
O próximo passo foi transformar este arquivo, em um arquivo na formatação igual ao grafo.txt pedido no enunciado do projeto.
Fizemos um código em C++ que transforma um arquivo como o descrito acima no formato desejado, o código também se encontra no github, mas por motivos de brevidade não será incluso 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;
}
```

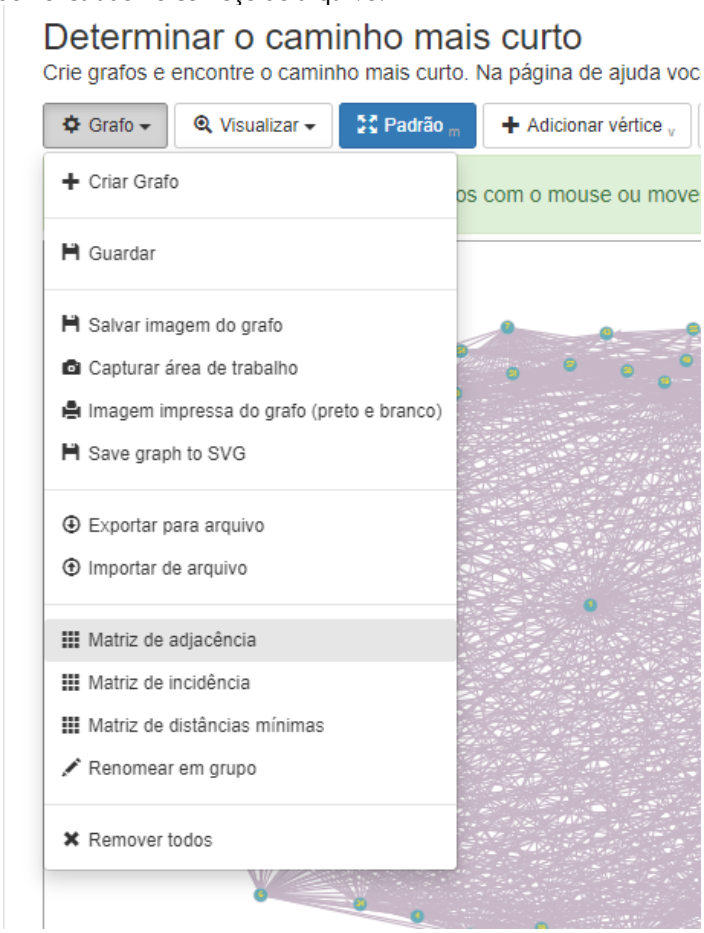
Após este passo, com a matriz de adjacência formada, basta imprimi-la no formato aceito pelo site GraphOnline, onde independente do peso, atribuímos 1 para onde há aresta e zero para onde não existam arestas. Foi desenvolvido o código abaixo:

```
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

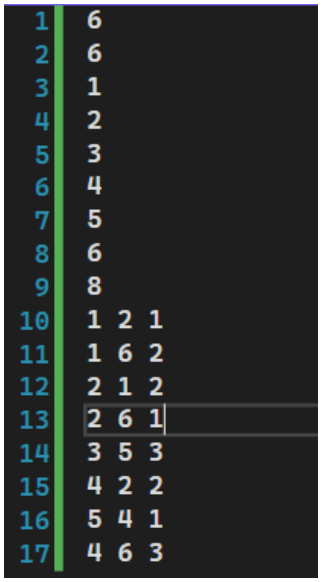
```
TGrafo& TGrafo::FileToGraph(std::string fileName)
{
    std::ifstream file(fileName);

    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:

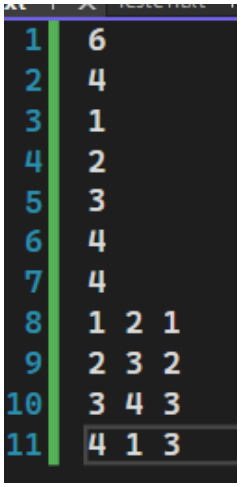
```
std::cout << "Teste ex 1:";
std::cout << "Lendo o arquivo e construindo matriz ... \n";
TGrafo grafo = grafo.FileToGraph("./Teste1.txt");
std::cout << "\nmatriz 1 obtida a partir de arquivo\n";
grafo.show();
std::cout << "_____________________________________________________________________________\n";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

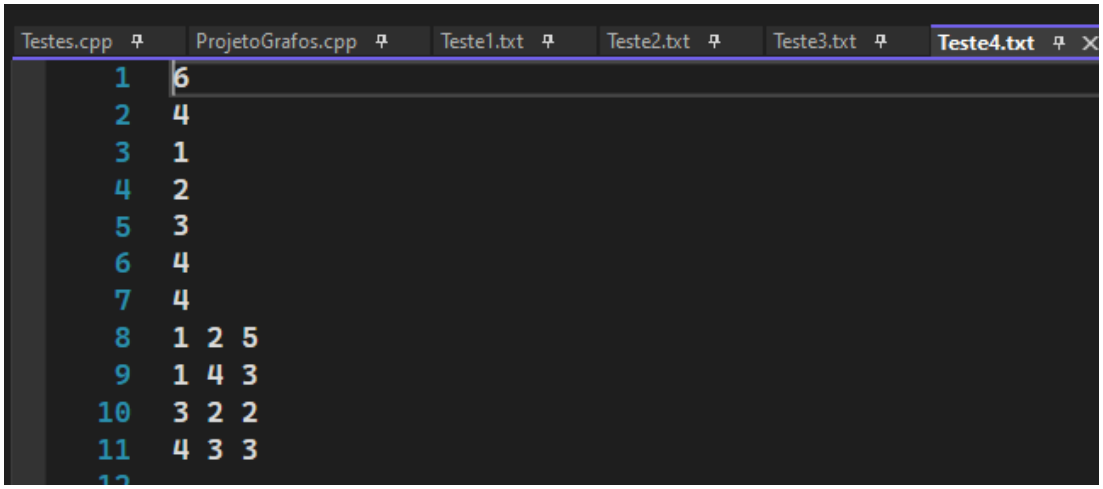
```
TGrafo grafo3(3);
grafo3.insereA(0, 1, 1);
grafo3.insereA(0, 2, 2);
grafo3.insereA(1, 2, 5);
grafo3.insereA(2, 1, 3);

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:

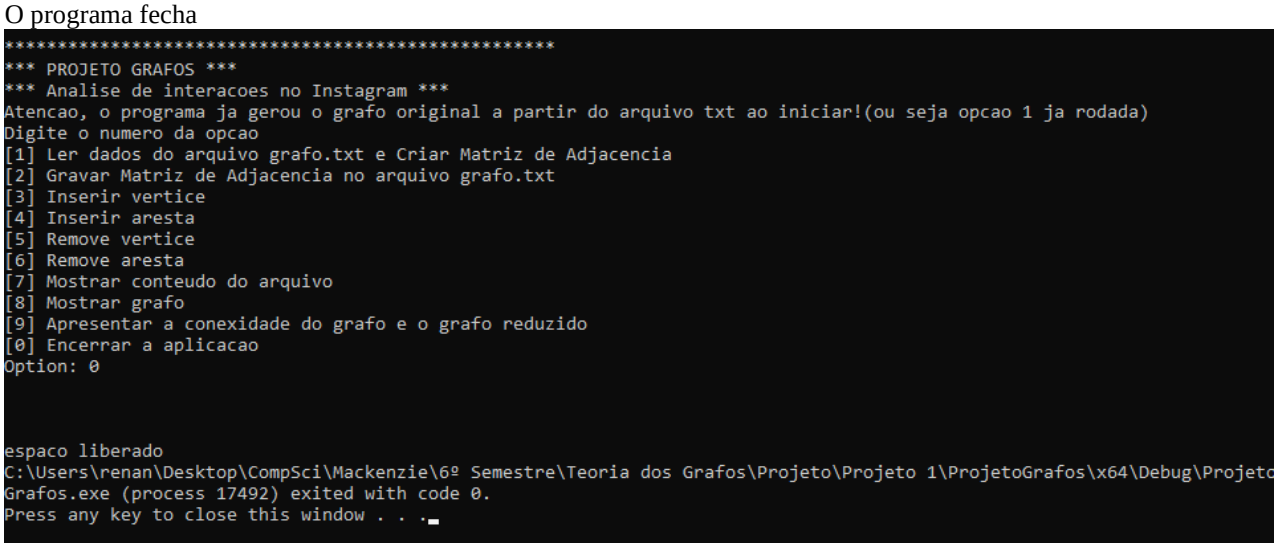
```
Grafo imputado
n: 4
m: 4

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



APENDICE:

LINK GITHUB:

https://github.com/RenanTagliaferro/Projeto1Grafos

**FAVOR LEIA O README.