TP_GRAFO

Programa código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Collections;
using System.Diagnostics;

namespace Grafo_2022_2
{
   class Programa
   {
      //Menu para chamar os metodos a serem executados pelo programa
      static int menu()
      {
         Console.Clear();
         Console.WriteLine("0 - preencher grafo aleatoriamente para testes"); //funcionava
         Console.WriteLine("1 - Exibir grafo");
         Console.WriteLine("2 - Incluir vértice");
         Console.WriteLine("3 - Incluir aresta");
         Console.WriteLine("4 - Remover vértice");
         Console.WriteLine("5 - Remover aresta");
         Console.WriteLine("6 - Testar adjacência");
         Console.WriteLine("7 - Verificar completo");
         Console.WriteLine("8 - Verificar totalmente desconexo");
         Console.WriteLine("9 - Verificar euleriano");
```

```
    Console.WriteLine("10 - Reiniciar grafo");
    Console.WriteLine("11 - Colorir grafo");
    Console.WriteLine("12 - dijkstra");
    Console.WriteLine("13 - Salvar o Grafo Gerado");
    Console.WriteLine("14 - Sair");
    Console.Write("Opção Desejada: ");

    return (int.Parse(Console.ReadLine()));
}
//Inicio programa Main
static void Main(string[] args)
{
    //Principal para o programa
    {
        //string de id dos vertices.
        string Id_Name1, Id_Name2;
        //ints para chamar setar peso para as arestas e vertices
        int Menu, V1;
        Grafo G = new Grafo();
        Algoritmo_1 A1 = new Algoritmo_1();
        Colorir_Grafo C = new Colorir_Grafo();
        NAIVE nAive = new NAIVE();
        SalvaGrafo salvaGrafo = new SalvaGrafo();
        do
        {
            Menu = menu();

            switch (Menu)
            {
```

```
//tentativa de automatizar um grafo para teste
case 0:
   Console.Clear();
   G.CriarGrafoTeste();
   Console.WriteLine("10 Vertices para teste criados");
   Console.WriteLine("Aperte qualquer tecla para continuar");
   Console.ReadKey();
   break;

case 1: // exibir grafo gerado
   Console.Clear();
   G.exibirGrafo();
   Console.ReadKey();
   break;

case 2: // incluir vertice - nome e peso
   Console.Clear();
   Console.Write("Informe o identificador do vértice: ");
   Id_Name1 = Console.ReadLine();
   Console.WriteLine("Informe seu peso");
   V1 = int.Parse(Console.ReadLine());
   G.incluirVertice(Id_Name1, V1);
   Console.ReadKey();
   //G.preencheGrafo();
   break;

case 3: // incluir aresta a partir do vertice informado
   Console.Clear();
   Console.Write("Informe vértice de origem: ");
```

```
    Id_Name1 = Console.ReadLine();
    Console.Write("Informe vértice de destino: ");
    Id_Name2 = Console.ReadLine();
    Console.WriteLine("Informe o peso da aresta a ser inserida");
    V1 = int.Parse(Console.ReadLine());
    //Sempre validar se existe adjacencia já entre os vertices
    if (G.adjacentes(Id_Name1, Id_Name2))
       Console.WriteLine("Os vértices {0} e {1} já são adjacentes.", Id_Name1, Id_Name2);
    else
       //se não houver inclui as arestas
       G.incluirAresta(Id_Name1, Id_Name2,V1);
    Console.ReadKey();
    break;

 case 4: // remover vertice
    Console.Clear();
    Console.Write("Informe o identificador do vértice: ");
    Id_Name1 = Console.ReadLine();
    G.removerVertice(Id_Name1);
    Console.ReadKey();
    break;

 case 5: // remover aresta
    Console.Clear();
    Console.Write("Informe vértice de origem: ");
    Id_Name1 = Console.ReadLine();
    Console.Write("Informe vértice de destino: ");
    Id_Name2 = Console.ReadLine();
    G.removerAresta(Id_Name1, Id_Name2);
```

```
            Console.ReadKey();
            break;

        case 6: // testar adjacência
            Console.Clear();
            Console.Write("Informe vértice de origem: ");
            Id_Name1 = Console.ReadLine();
            Console.Write("Informe vértice de destino: ");
            Id_Name2 = Console.ReadLine();
            if (G.adjacentes(Id_Name1, Id_Name2))
                Console.WriteLine("Os vértices {0} e {1} são adjacentes.", Id_Name1, Id_Name2);
            else
                Console.WriteLine("Os vértices {0} e {1} não são adjacentes.", Id_Name1, Id_Name2);
            Console.ReadKey();
            break;

        case 7: // verificar grafo completo
            Console.Clear();
            if (G.vazio())
                Console.WriteLine("Grafo vazio.");
            else if (G.completo())
                Console.Write("O grafo é completo.");
            else
                Console.WriteLine("O grafo não é completo.");
            Console.ReadKey();
            break;

        case 8: // verificar grafo desconexo
            Console.Clear();
```

```
    if (G.vazio())
        Console.WriteLine("Grafo vazio.");
    else if (G.totalmenteDesconexo())
        Console.Write("O grafo é totalmente desconexo.");
    else
        Console.WriteLine("O grafo não é totalmente desconexo.");
    Console.ReadKey();
    break;

case 9: // verificar euleriano tentar
    Console.Clear();/*
    if (G.euleriano())
        Console.Write("O grafo é euleriano.");
    else
        Console.WriteLine("O grafo não é euleriano.");
    Console.ReadKey();*/
    break;

case 10: // reiniciar grafo
    Console.WriteLine("Você deseja salvar o grafo? \n1 - Sim\n2 - Não");
    int Opcao = int.Parse(Console.ReadLine());
    do
    {
        switch (Opcao)
        {
            case 1:
                salvaGrafo.SalvarGrafo(G);
                break;
            case 2:
```

```
                    G.ReiniciarGrafo();
                    break;
                }
            } while (true);

        case 11: //colorir os vertices
            Console.Clear();
            C.colorirGrafo(G);
            Console.WriteLine("Grafo colorido");
            Console.ReadKey();
            break;

        case 12: // algoritmo de Dijkstra - tentativa - ref: https://eximia.co/o-algoritmo-de-dijkstra-em-c/
            Console.Clear();
            Console.WriteLine("Algoritmo de Djkistra");

            Console.Write("Informe vértice de origem: ");
            Id_Name1 = Console.ReadLine();
            Console.Write("Informe vértice de destino: ");
            Id_Name2 = Console.ReadLine();
            //passar o grafo e fazer um caminho de Djikstra entre os vertices
            A1.A_dijkstra(G, Id_Name1, Id_Name2);
            Console.ReadKey();
            break;

        case 13: //metodo de salvar arquivo com todos os dados do grafo. sobrescreve o arquivo já gerado. não consegui gerar versões: (1)(2)
            Console.Clear();
```

```
                    salvaGrafo.SalvarGrafo(G);
                    //SalvarGrafo();
                    Console.ReadKey();
                    break;
                case 14: //metodo para tentar usar NAIVA- TARJAN
// tests simple model presented on
https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm
                    Console.Clear();
                    Console.WriteLine("Algoritmo de Tarjan");
                    //recebe a lista de vertices para testar, entendi que é assim
                    var lista_ciclo = nAive.DetectaCiclo(G.vertices);
                    Console.ReadKey();
                    break;
            }
        } while (Menu != 15);
    }
  }


 }
}
```

GRAFO CÓDIGO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

//gerar o grafo em si e suas operações
namespace Grafo_2022_2
{
   //Gerar os vertices do Grafo - objeto
   public class vertice
   {
      //gerar um objeto Vertice com os atributos
      public string Id_Vertice;
      public int Id_Peso = 0;
      public int Id_Cor = 99;

      //gerar uma lista de adjacencias do vertice
      public List<aresta> adjacencias = new List<aresta>();

      //Usar para Djikstra
      public int rotulo = 0; //rotular o vertice
      public bool permanente = false;

      //pega o atributo do valor da aresta e aponta para seu destino
      public class aresta
      {
```

```
        //Pega e atribui um valor na aresta
        public string Id_Destino; //identifica a vertice destino - não orientado
        public int Peso_Distancia; //gera um valor entre os vertices - da um peso para a aresta


    }
}
public class Grafo
{
    //Lista teste utilizado para o metodo de teste de funcionamento dos metodos gerados, não utilizada mais
    public List<vertice> Teste_vertices = new List<vertice>();
    //função para realizar testes, total convicção que funciona
    public void preencheGrafo()
    {
        vertice v;
        for (int i = 1; i < 4; i++)
        {
            v = new vertice();
            v.Id_Vertice = i.ToString(); //nomeia
            v.Id_Peso = i; //da peso pro vertice
            Teste_vertices.Add(v); //adiciona na lista
            numVertices++;
            if (i == 2)
            {
                incluirAresta(1.ToString(), 2.ToString(), 100);
                adjacentes(1.ToString(), 2.ToString());
            }
        }
```

```
}
//função para realizar testes
public void PreencherVertices()//função para realizar testes
{
   vertice v;
   for (int i = 0; i < 10; i++)
   {
      v = new vertice();
      v.Id_Vertice = i.ToString();
      v.Id_Peso = i;
      Teste_vertices.Add(v);
      vertices.Add(v);
      numVertices++;
   }
}
//função para realizar testes
public void CriarGrafoTeste()
{
   PreencherVertices();
}
//Gerar uma lista com os vertices e seus pesos e preencher a matriz com os vertices.
public List<vertice> vertices = new List<vertice>();
//Gerar um contador de vertices no Grafo
public int numVertices = 0;
//Exibir o Grafo gerado
public void exibirGrafo()
{
   //Valida a quantidade de vertices, inciando em 0.
   Console.WriteLine("Grafo possui {0} vértices. \n\n", numVertices);
```

```
        //informa todos os vertices e seus atributos
        foreach (vertice v in vertices)
        {
            //Gerar informação do identificador do vertice, seu peso e sua cor
            Console.Write("Vértice {0}, valor: {1}, cor: {2} é adjacente a: ", v.Id_Vertice, v.Id_Peso,
v.Id_Cor);
            Console.WriteLine();
            //Gerar um relatório de quais os vertices que eles alcançam - ao menos deveria
            foreach (vertice.aresta V_aux in v.adjacencias)
                Console.Write("Vertice adjacente: \n", V_aux.Id_Destino);
        }
    }
    //verificar por nome do vertice
    public vertice existeVertice(string n)
    {
        vertice Aux = null;
        int i;

        i = 0;
        //Corre a lista e verifica nome a nome a existencia do vértice
        while ((i < vertices.Count) && (vertices.ElementAt(i).Id_Vertice != n))

            i++;

        if (i == vertices.Count) //se não tiver retorno, o metodo fica vazio
            return Aux = null;
        else //se houver algum retorno, ele volta o vertice instanciado.
            Aux = vertices.ElementAt(i);
```

```
    return Aux;
}
//Adicionar vertices
public void incluirVertice(string _Name, int _Peso)
{
    //vertice auxiliar
    vertice v = new vertice();
    v.Id_Vertice = _Name;
    v.Id_Peso = _Peso;

    //envia o vertice que está sendo inserido no sistema
    if (existeVertice(v.Id_Vertice) != null)
        Console.WriteLine("Vértice {0} já existe no Grafo.", v.Id_Vertice);
    else //após a validação se não existir ele é inserido no sistema
    {
        //Adiciona o vertice na lista
        vertices.Add(v);
        numVertices++;
    }
}
//adicionar arestas não orientadas - ao menos deveria
public void incluirAresta(string V_a1, string V_a2, int Peso_Distancia)
{
    vertice VerAux1, VerAux2; //dois vertices auxiliares para instanciar
    vertice.aresta Va; //instaciar a aresta do vertice
    int i;
    //Valida se existe os vertices no grafo
    if (existeVertice(V_a1) != null || existeVertice(V_a2) != null)
        Console.WriteLine("Um dos dois vértices ({0} ou {1}) não existe no grafo.", V_a1, V_a2);
```

```
else
{
   if (adjacentes(V_a1, V_a2))
      Console.WriteLine("{0} e {1} já são adjacentes no grafo.", V_a1, V_a2);
   else
   {
      //setar as vertices auxiliares ponta a ponta na lista de arestas
      //vertice inicial
      VerAux1 = existeVertice(V_a1);
      //vertice final
      VerAux2 = existeVertice(V_a2);

      //gerar a aresta e apontar seu Id_Destino
      Va = new vertice.aresta();
      Va.Id_Destino = VerAux2.Id_Vertice;
      i = 0;
      while ((i < vertices.Count) && (vertices.ElementAt(i).Id_Vertice != VerAux1.Id_Vertice))
         i++;
      //Gera aresta do vertice 1 -> 2 e adiciona como objeto
      vertices.ElementAt(i).adjacencias.Add(Va);
      //gerar a aresta e apontar seu Id_Destino
      Va = new vertice.aresta();
      Va.Id_Destino = VerAux1.Id_Vertice;
      i = 0;

      while ((i < vertices.Count) && (vertices.ElementAt(i).Id_Vertice != VerAux2.Id_Vertice))
         i++;
      //Gera aresta do vertice 2 -> 1 e adiciona como objeto
      vertices.ElementAt(i).adjacencias.Add(Va);
```

```
            }
        }
    }
    //Valida se já existe a ajacencia entre os vertices
    public bool adjacentes(string V_a1, string V_a2)
    {
        vertice v;
        int i;

        //primeiro validar a existencia dos vertices
        if (existeVertice(V_a1) != null && existeVertice(V_a2) != null)
        {
            i = 0;
            //varrer a lista de vertices e encontrar o vertice inicial
            while (vertices.ElementAt(i).Id_Vertice != V_a1)
                i++;

            v = vertices.ElementAt(i);

            i = 0;

            //varrer a lista de vertices e encontrar o vertice inicial e validar se encontra alguma adjacencia entre eles
            while ((i < v.adjacencias.Count) && (v.adjacencias.ElementAt(i).Id_Destino != V_a2))
                i++;

            //se rodar toda a lista e não encotrar
            if (i == v.adjacencias.Count)
                return (false);
```

```
            //se rodar toda a lista e encotrar
            else
                return (true);
        }
        else
            //primeiro validar a existencia dos vertices e se não existir finaliza
            return (false);
    }
    public int posicaoVertice(string V_a1)
    {
        int i = 0;
        //validar a existencia do vertice e suas adjacencias

        //verifica a existencia de vertices e valida todas suas adjacencias
        while ((existeVertice(V_a1) != null) && (i < existeVertice(V_a1).adjacencias.Count) &&
(vertices.ElementAt(i).Id_Vertice != existeVertice(V_a1).Id_Vertice))
            i++;

        if (i == vertices.Count)
            return (-1);
        else
            return (i);

    }
    //pega a lista de adjacencia e valida se existe o vertice e gera a sequencia de posição ao vertice 1 ->
2
    public int posicaoAresta(List<vertice.aresta> adjacencias, string V_a1)
    {
        int i = 0;
```

```
        //validar a existencia do vertice e suas arestas
        while ((existeVertice(V_a1) != null) && (i < adjacencias.Count) &&
(adjacencias.ElementAt(i).Id_Destino != V_a1))
            i++;

        if (i == adjacencias.Count)
            return (-1);
        else
            return (i);

    }
    //remover o vertice
    public void removerVertice(string V_a1)
    {
        int i;
        //validar a existencia do vertice no Grafo
        if (existeVertice(V_a1) != null)
            Console.WriteLine("O vértice {0} não existe no Grafo.", V_a1);
        else
        {
            foreach (vertice V_a2 in vertices)
            {
                //valida as adjacencias de vertices
                i = posicaoAresta(V_a2.adjacencias, V_a1);
                if (i != -1)
                    //valida as arestas e remove
                    removerAresta(V_a1, V_a2.Id_Vertice);
            }
```

```
        //remove vertices na lista corretamente
        vertices.RemoveAt(posicaoVertice(V_a1));
        numVertices--;
    }
}
//Remover aresta
public void removerAresta(string V_a1, string V_a2)
{
    //auxiliar os posicionamentos dos vertices e remover as arestas corretamente
    int i, j;

    if (!adjacentes(V_a1, V_a2))
        Console.WriteLine("Os vértices {0} e {1} não são adjacentes.", V_a1, V_a2);
    else
    {
        i = posicaoVertice(V_a1);
        j = posicaoAresta(vertices.ElementAt(i).adjacencias, V_a2);
        vertices.ElementAt(i).adjacencias.RemoveAt(j);

        i = posicaoVertice(V_a2);
        j = posicaoAresta(vertices.ElementAt(i).adjacencias, V_a1);
        vertices.ElementAt(i).adjacencias.RemoveAt(j);
    }
}
//Validar se o Grafo é completo
public bool completo()
{
    foreach (vertice v in vertices)
        if (v.adjacencias.Count < (numVertices - 1))
```

```
                return (false);


            return (true);
        }
        //Validar se o Grafo é completamente desconexo
        public bool totalmenteDesconexo()
        {
            foreach (vertice v in vertices)
                if (v.adjacencias.Count != 0)
                    return (false);


            return (true);
        }
        //Validar se o Grafo existe
        public bool vazio()
        {
            return (numVertices == 0);
        }
        public void ReiniciarGrafo()
        {
            vertices.Clear();
            numVertices = 0;
        }
    }
}
```

COLORIR GRAFO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Grafo_2022_2
{
    class Colorir_Grafo
    {
        /*tentativa de colorir o grafo - referenciado código usando C++:
https://acervolima.com/minimize-o-custo-para-colorir-todos-os-vertices-de-um-grafico-nao-
direcionado-usando-determinada-operacao/ - ideia baseada naquela parada do slide Grafo
Adjunto (ou Grafo de Linha) do slide (22-Coloração - Guloso - Welsh-Powell.pptx)*/
        //iria tentar fazer uma inteface no visual basic para tentar fazer alguma coisa
com cores hexadecimais, mas não deu certo e ai so atribui numero
        //iria tentar usar: https://www.macoratti.net/18/10/c_hexacor1.htm - para validar
de acordo com a numeração de hexadecimal
        public void colorirGrafo(Grafo G)
        {
            //vetor/lista de vertices
            int[] cores = new int[G.vertices.Count];
            List<vertice> adjacentes;
            int aux = 0;
            int opc;

            //vai "colorir" com números
            for (int i = 0; i < cores.Length; i++)
            {
                cores[i] = i + 1;
            }

            Console.WriteLine("Digite a opção de coloração: \n- Pior solução = 1\n-
Guloso = 2");
            opc = int.Parse(Console.ReadLine());
            switch (opc)
            {
                //colore todos os vertices de uma cor
                case 1:
                    foreach (vertice v in G.vertices)
                    {
                        v.Id_Cor = cores[aux];
                        aux++;
                    }
                    break;
                //algoritmo guloso, atribui em sequencia as cores verificando apenas se
ela não está sendo utilizada por um adjacente
                case 2:
                    G.vertices[0].Id_Cor = cores[0];
                    for (int j = 1; j < G.vertices.Count; j++)//para cada vertice
                    {
                        adjacentes = new List<vertice>();
                        foreach (vertice.aresta a in G.vertices[j].adjacencias)//para
cada adjacente ao vertice j
                        {
```

```
                            for (int k = 0; k < G.vertices.Count; k++)//para cada vertice
na lista de vertices
                            {
                                if (a.Id_Destino == G.vertices[k].Id_Vertice)//verifica
se o vertice na lista é adj ao vertice j
                                {
                                    adjacentes.Add(G.vertices[k]); //adiciona a lista os
vertices adj ao j
                                }
                            }
                        }

                        Console.WriteLine("vertice {0} tem {1} adjacentes",
G.vertices[j].Id_Peso, adjacentes.Count);

                        for (int i = 0; i < cores.Length; i++)//para cada cor
                        {
                            if (G.vertices[j].Id_Cor == 0)
                            {
                                int contador = 0;
                                for (int k = 0; k < adjacentes.Count; k++)//para cada
adjacente de j
                                {
                                    if (adjacentes[k].Id_Cor != cores[i])//se a cor não
esta sendo utilizada por um adjacente
                                    {
                                        contador++;
                                    }
                                }
                                if (contador == adjacentes.Count)//se a cor não está
sendo utilizada por nenhum dos adjacentes
                                {
                                    Console.WriteLine("colorindo {0} de {1}",
G.vertices[j].Id_Peso, cores[i]);
                                    G.vertices[j].Id_Cor = cores[i];//colore o vertice
com aquela cor;
                                }
                            }
                        }
                    }
                    break;
            }

        }
    }
}
```

DIJKSTRA

```
public class Algoritmo_1
    {
        //tentativa de fazer uma validação de Dijkstra
        //ref: https://www.revista-programar.info/artigos/algoritmo-de-dijkstra/
        public void A_dijkstra(Grafo G, string Id_Name1, string Id_Name2)
        {
            List<vertice> caminhoAtual = new List<vertice>(), caminho = new
List<vertice>();
            /*vertice verticedestino = new vertice();
            vertice verticeatual = new vertice();*/
            vertice atual;
            //setando os vertices de acordo com seu tipo
            foreach (vertice v in G.vertices)
            {
                if (v.Id_Vertice == G.existeVertice(Id_Name1).Id_Vertice)
                {
                    v.permanente = true;
                    v.rotulo = 0;
                    atual = v;
                    caminhoAtual.Add(v);
                }
                else
                {
                    v.permanente = false;
                    v.rotulo = int.MaxValue;
                }
            }

            foreach (vertice v in G.vertices)
            {
                rotular(v,G);
            }

            foreach (vertice v in G.vertices)
            {
                if (v.Id_Vertice == G.existeVertice(Id_Name2).Id_Vertice)
                {
                    Console.WriteLine("id da origem: {0}\nrotulo da origem: {1}",
v.Id_Vertice, v.rotulo);
                }
                else if (v.Id_Vertice == G.existeVertice(Id_Name2).Id_Vertice)
                {
                    Console.WriteLine("d do destino: {0}\nrotulo do destino: {1}",
v.Id_Vertice, v.rotulo);
                }
            }
        }
        //atribuir o menor caminho entre vertices
        public void rotular(vertice v, Grafo G)
        {
            foreach (vertice comparador in G.vertices)
            {
```

```
                foreach (vertice.aresta a in v.adjacencias)
                {
                    if (G.adjacentes(a.Id_Destino, v.Id_Vertice))
                    {
                        if (a.Peso_Distancia < v.rotulo)
                        {
                            v.rotulo = a.Peso_Distancia;
                        }
                    }
                }
            }
            v.permanente = true;
        }
    }
}
```

SALVAR GRAFO

```
class SalvaGrafo : Grafo
    {
        //Salvar todo o grafo gerado - ref: https://learn.microsoft.com/en-
us/dotnet/csharp/programming-guide/concepts/linq/how-to-compute-column-values-in-a-csv-
text-file-linq / https://www.youtube.com/watch?v=Jxf9Klwdefc

        //tentativa com o modelo HERO
        /*
         public class Hero
{
    public Hero(string name, string phone, DateTime birthDate)
    {
        Name = name;
        Phone = phone;
        BirthDate = birthDate;
    }

    public string Name { get; set; }
    public string Phone { get; set; }
    public DateTime BirthDate { get; set; }

    public static implicit operator string(Hero hero)
        => $"{hero.Name},{hero.Phone},{hero.BirthDate.ToString("yyyy-MM-dd")}";

    public static implicit operator Hero(string line)
    {
        var data = line.Split(",");
        return new Hero(
            data[0],
            data[1],
            data[2].ToDateTime());
    }
}
         */
        public string IdVertice { get; set; }
        public int PesoVertice { get; set; }
        public int CorVertice { get; set; }
        public string AdjacenciaVertice { get; set; }
        public string VerticeDestino { get; set; }
        public int PesoDestino { get; set; }


        //um operador para transformar o vertice em uma string - deveria ao menos
        public static implicit operator string(SalvaGrafo salva)
            =>
$"{salva.IdVertice},{salva.PesoVertice},{salva.CorVertice},{salva.AdjacenciaVertice},{sal
va.VerticeDestino},{salva.PesoVertice}";


        public void SalvarGrafo(Grafo _grafo)
        {
            //criar uma pasta no c:
```

```
            string Pasta_Grafo = @"C:\Grafo_Heron_2022\dados/";

            if (!Directory.Exists(Pasta_Grafo))
                Directory.CreateDirectory(Pasta_Grafo);

            //roda a lista de vertices
            /* foreach (var V_aux in _grafo.vertices)
             {
                 Console.WriteLine(V_aux);
             }*/
            //transformar os vertices para string
            //gera a data para ser gravada em linhas
            //pega a informação (objeto) a ser tranformada
            //pega a lista de _grafo.vertice e transforma suas informações
            //após selecionar pega o objeto e tranforma para string
            //IEnumerable<Grafo> data = _grafo.vertices.Any(x => (string) x.ToString());

            // string data = strings.

            //File.WriteAllTextAsync(Path.Combine(Pasta_Grafo, "Grafo.csv",
data.ToString());


            //criar uma pasta no c:
            //string Pasta_Grafo = @"C:\Grafo_Heron_2022\dados/";

            if (!Directory.Exists(Pasta_Grafo))
                Directory.CreateDirectory(Pasta_Grafo);
            //string[] teste = File.ReadAllLines(Path.Combine(Pasta_Grafo, "Grafo.csv"));
            //gerar o arquivo CSV na pasta grafo em C:
            using StreamWriter Salvar_Grafo_Gerado = new
StreamWriter(Path.Combine(Pasta_Grafo, "Grafo.csv"));
            //using StreamWriter Salvar_Grafo_Gerado2 = new
StreamWriter((@"C:\Grafo_Heron_2022dados//Grafo.txt"), true, Encoding.ASCII);

            //salvar as informações do objeto vertice
            foreach (vertice v in vertices)
            {
                Salvar_Grafo_Gerado.Write("Teste id vertice: ", v.Id_Vertice);
                Salvar_Grafo_Gerado.WriteLine("Teste peso vertice: ", v.Id_Peso);
                Salvar_Grafo_Gerado.WriteLine("Teste cor vertice: ", v.Id_Cor); //csv
                foreach (vertice.aresta A_Aux in v.adjacencias)
                {
                    Salvar_Grafo_Gerado.WriteLine("Teste id vertice destino: ",
A_Aux.Id_Destino);
                    Salvar_Grafo_Gerado.WriteLine("Teste peso vertice destino: ",
A_Aux.Peso_Distancia);
                }
                // Salvar_Grafo_Gerado2.Equals(v); //txt
                // Salvar_Grafo_Gerado.Close();
                //salva as informações de arestas
            }

            Console.WriteLine("Arquivo Salvo");
            Console.WriteLine("Aperte qualquer tecla para continuar");
            Console.ReadKey();
```

```
            }
        }
    }
```

SALVA GRAFO 2

```
public class TESTEsSalvar
    {
        /*
        //criar uma pasta no c:
        string Pasta_Grafo = @"C:\Grafo_Heron_2022\dados/";

        if (!Directory.Exists(Pasta_Grafo))
            Directory.CreateDirectory(Pasta_Grafo);
        //string[] teste = File.ReadAllLines(Path.Combine(Pasta_Grafo, "Grafo.csv"));
        //gerar o arquivo CSV na pasta grafo em C:
        using StreamWriter Salvar_Grafo_Gerado = new
StreamWriter(Path.Combine(Pasta_Grafo, "Grafo.csv"));
        //using StreamWriter Salvar_Grafo_Gerado2 = new
StreamWriter((@"C:\Grafo_Heron_2022dados//Grafo.txt"), true, Encoding.ASCII);

        //salvar as informações do objeto vertice
        foreach (vertice v in vertices)
        {
            Salvar_Grafo_Gerado.Write("Teste id vertice: ", v.Id_Vertice);
            Salvar_Grafo_Gerado.WriteLine("Teste peso vertice: ", v.Id_Peso);
            Salvar_Grafo_Gerado.WriteLine("Teste cor vertice: ", v.Id_Cor); //csv
            foreach (vertice.aresta A_Aux in v.adjacencias)
            {
                Salvar_Grafo_Gerado.WriteLine("Teste id vertice destino: ",
A_Aux.Id_Destino);
                Salvar_Grafo_Gerado.WriteLine("Teste peso vertice destino: ",
A_Aux.Peso_Distancia);
            }
            // Salvar_Grafo_Gerado2.Equals(v); //txt
            // Salvar_Grafo_Gerado.Close();
            //salva as informações de arestas
        }
        */
        class Hero
        {
            public string IdVertice { get; set; }
            public string PesoVertice { get; set; }
            public string BirthDate { get; set; }
        }
    }
}
```

NAÏVE – TARJAN

```
class NAIVE : Grafo
    /*
            ref: https://codeforces.com/blog/entry/71146?f0a28=2

            c++ code:

            int dfsAP(int u, int p) {
                int children = 0;
                low[u] = disc[u] = ++Time;
                for (int & v : adj[u]) {
                if (v == p) continue; // we don't want to go back through the same path.
                                      // if we go back is because we found another way
back
                if (!disc[v]) { // if V has not been discovered before
                    children++;
                    dfsAP(v, u); // recursive DFS call
                    if (disc[u] <= low[v]) // condition #1
                        ap[u] = 1;
                    low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
                } else // if v was already discovered means that we found an ancestor
                    low[u] = min(low[u], disc[v]); // finds the ancestor with the least
discovery time
            }
            return children;
        }

        void AP() {
            ap = low = disc = vector<int>(adj.size());
            Time = 0;
            for (int u = 0; u < adj.size(); u++)
                if (!disc[u])
                    ap[u] = dfsAP(u, u) > 1; // condition #2
        }
         */
    //validar se o grafo é completo
    //https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm
    {
        protected List<List<vertice>> CompAltConect;
        protected Stack<vertice> Stackar;
        private int APeso;

        public List<List<vertice>> DetectaCiclo(List<vertice> nodes)
        {
            CompAltConect = new List<List<vertice>>();
            foreach (vertice vAux in nodes)
            {
                if (vAux.Id_Peso < 0 )
                {
                    VerCompAltConect(vAux);
                }
            }
            return CompAltConect;
        }
        private void VerCompAltConect(vertice v)
```

```
        {
            v.Id_Peso = APeso;
            v.rotulo = APeso;

            APeso++;
            Stackar.Push(v);

            foreach (vertice.aresta A_Aux in v.adjacencias)
            {
                A_Aux.Peso_Distancia = APeso;
            }
            APeso++;
            Stackar.Push(v);

            foreach (vertice.aresta vAux in v.adjacencias)
            {
                if (vAux.Peso_Distancia < 0)
                {
                    VerCompAltConect(v);
                    v.rotulo = Math.Min(v.Id_Peso, vAux.Peso_Distancia);
                }
                else if (Stackar.Contains(v))
                {
                    v.Id_Peso = Math.Min(v.Id_Peso, vAux.Peso_Distancia);
                }
            }

            if (v.Id_Peso == v.rotulo)
            {
                List<vertice> cycle = new List<vertice>();
                vertice w;

                do
                {
                    w = Stackar.Pop();
                    cycle.Add(w);
                } while (v != w);

                CompAltConect.Add(cycle);
            }
        }
    }
}
```