INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE Campus Passo Fundo

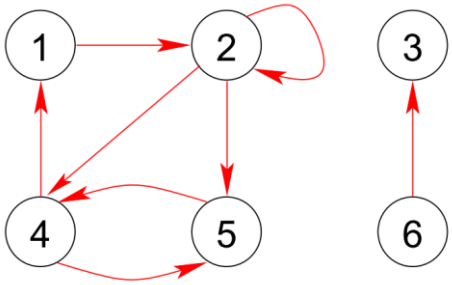
Ciência da Computação

Disciplina: Estrutura de Dados II

Professor: Adilso Nunes de Souza

Lista de exercícios 6

1 – Analise o dígrafo apresentado abaixo:



Em seguida apresente:

- Matriz de adjacência deste dígrafo:

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

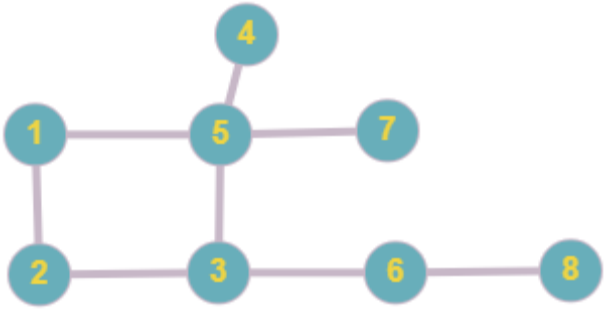
- Lista de adjacência deste dígrafo:

1 → 2
2 → 2 → 4 → 5
3
4 → 1 → 5
5 → 4
6 → 3

- Grau de todos os vértices do dígrafo:

Vértice	**Grau de Entrada**	**Grau de Saída**
1	1	1
2	2	3
3	1	0
4	2	2
5	2	1
6	0	1

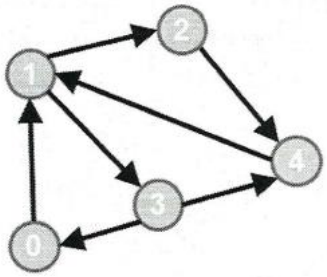
2 – Dado os vértices "V" e as arestas "A", desenhe o grafo correspondente:
V = {1, 2, 3, 4, 5, 6, 7, 8}
A = {(1, 2), (1, 5), (2, 3), (3, 5), (3, 6), (4, 5), (5,7), (6,8)}



3 – Dado o grafo apresentado abaixo, considerando como vértice inicial o 0 "zero" indique os possíveis caminhos para alcançar o vértice 4. Apresente este grafo em uma matriz de adjacência.

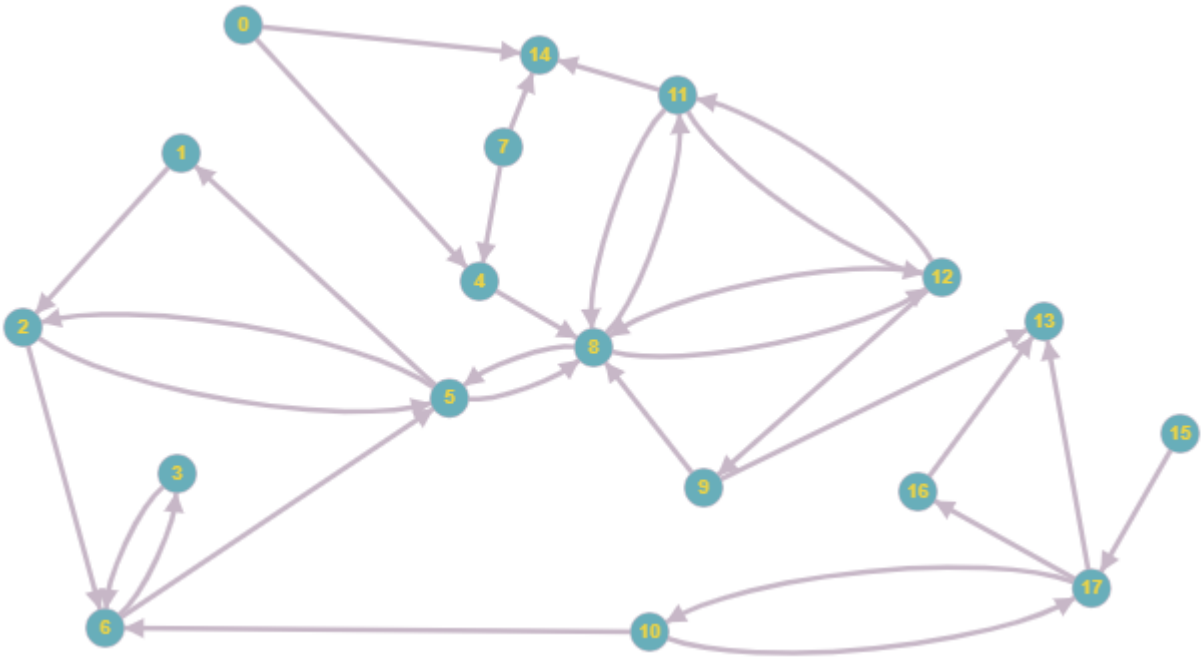
Possíveis caminho do vértice 0 para o vétice 4:
0 → 1 → 2 → 4
0 → 1 → 3 → 4

	0	**1**	**2**	**3**	**4**
0	0	1	0	0	0
1	0	0	1	1	0
2	0	0	0	0	1
3	1	0	0	0	1
4	0	1	0	0	0



4 – Dado a matriz de adjacência escreva o grafo correspondente.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0					1										1			
1			1															
2						1	1											
3							1											
4									1									
5		1	1						1									
6				1		1												
7					1										1			
8						1						1	1					
9									1					1				
10							1											1
11									1				1		1			
12									1	1		1						
13																		
14																		
15																		1
16														1				
17											1			1			1	



5 – Implemente um programa que manipule um grafo utilizando uma matriz de adjacência, sendo possível no máximo 20 vértices. O programa deverá possibilitar as seguintes funcionalidades:
0 – Sair
1 – Incluir aresta (deve informar o vértice inicial e o vértice final)
2 – Mostrar a Matriz na tela
3 – Remover aresta (deve informar o vértice inicial e o vértice final)
4 – Informado um determinado vértice deve apresentar quais são os seus vértices adjacentes.
5 – Informado um determinado vértice deve apresentar o grau deste vértice.

OBS: as opções 2, 3, 4, 5 só poderão ser acionadas após ter sido inserido ao menos uma aresta no grafo.

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#define tam 20

using namespace std;
void cria_matriz(int m[][tam]);
void mostra_matriz(int m[][tam]);
void incluir_aresta(int m[][tam]);
void remover_aresta(int m[][tam]);
void mostra_adjacente(int m[][tam]);
void mostra_grau(int m[][tam]);

main()
{
   int menu;
   int mat[tam][tam];
   cria_matriz(mat);

   do{
      system("cls");
      cout << "MENU DE OPCOES" << endl;
      cout << "0 - Sair" << endl;
      cout << "1 - Incluir aresta" << endl;
      cout << "2 - Mostrar a Matriz" << endl;
      cout << "3 - Remover aresta" << endl;
      cout << "4 - Mostra vertices adjacentes" << endl;
      cout << "5 - Mostra o grau de um vertice" << endl;
      cout << "Sua escolha: ";
      cin >> menu;
      fflush(stdin);
      switch(menu)
      {
        case 0:
          system("cls");
          cout << "Programa finalizado!";
          getchar();
          break;
        case 1:
          system("cls");
          incluir_aresta(mat);
          cout << "Aresta inserida com sucesso.";
          getchar();
          break;
        case 2:
          system("cls");
          mostra_matriz(mat);
          getchar();
          break;
        case 3:
          system("cls");
          remover_aresta(mat);
          getchar();
          break;
        case 4:
          system("cls");
          mostra_adjacente(mat);
          getchar();
```

```
            break;
         case 5:
            system("cls");
            mostra_grau(mat);
            getchar();
            break;
      };
   }while(menu != 0);
}

void cria_matriz(int m[][tam])
{
   for(int l = 0; l < tam; l++)
   {
      for(int c = 0; c < tam; c++)
         m[l][c] = 0;
   }
}

void mostra_matriz(int m[][tam])
{
   for(int l = 0; l <= tam; l++)
   {
      if(l < 9)
         cout << l  << "    ";
      else
         cout << l  << "   ";
   }
   cout << "\n";
   for(int l = 0; l < tam; l++)
   {
      if(l < 9)
         cout << l + 1 << "    ";
      else
         cout << l + 1 << "   ";

      for(int c = 0; c < tam; c++)
      {
           cout << m[l][c] << "    ";
      }
      cout << "\n";
   }
}

void incluir_aresta(int m[][tam])
{
   int ini, fim;
   cout << "Informe o vertice inicial: ";
   cin >> ini;
   fflush(stdin);
   cout << "Informe o vertice final: ";
   cin >> fim;
   fflush(stdin);
   m[ini - 1][fim - 1] = 1;
}

void remover_aresta(int m[][tam])
{
```

```
   int ini, fim;
   cout << "Informe o vertice inicial: ";
   cin >> ini;
   fflush(stdin);
   cout << "Informe o vertice final: ";
   cin >> fim;
   fflush(stdin);
   if(m[ini - 1][fim - 1] == 0)
     cout << "Aresta inexistente.";
   else
   {
     m[ini - 1][fim - 1] = 0;
     cout << "Aresta removida com sucesso.";
   }
}

void mostra_adjacente(int m[][tam])
{
   int v, cont = 0;
   cout << "Informe o vertice: ";
   cin >> v;
   fflush(stdin);
   cout << "\nAdjacentes ao vertice " << v << ": " << endl;
   for(int c = 0; c < tam; c++)
   {
      if(m[v - 1][c] == 1)
      {
        cout << c + 1 << ", ";
        cont++;
      }
   }
   if(cont == 0)
      cout << "Nenhum vertice adjacente encontrado";
}

void mostra_grau(int m[][tam])
{
   int v, cont = 0;
   cout << "Informe o vertice: ";
   cin >> v;
   fflush(stdin);
   cout << "\nGrau do vertice " << v << ": " << endl;
   for(int c = 0; c < tam; c++)
   {
      if(m[v - 1][c] == 1)
      {
        cont++;
      }
   }
   cout << cont;
}
```