

# TERCEIRO TESTE

Universidade Federal de Goiás (UFG) - Regional Jataí  
Bacharelado em Ciência da Computação  
Teoria de Grafos  
Esdras Lins Bispo Jr.

11 de julho de 2017

## ORIENTAÇÕES PARA A RESOLUÇÃO

- A avaliação é individual, sem consulta;
- A pontuação máxima desta avaliação é 10,0 (dez) pontos, sendo uma das 06 (seis) componentes que formarão a média final da disciplina: quatro testes, uma prova e os exercícios de aquecimento;
- A média final ( $MF$ ) será calculada assim como se segue

$$MF = MIN(10, S)$$
$$S = \left(\sum_{i=1}^4 0,2.T_i\right) + 0,2.P + 0,1.EA$$

em que

- $S$  é o somatório da pontuação de todas as avaliações,
  - $T_i$  é a pontuação obtida no teste  $i$ ,
  - $P$  é a pontuação obtida na prova, e
  - $EA$  é a pontuação total dos exercícios de aquecimento.
- O conteúdo exigido compreende os seguintes pontos apresentados no Plano de Ensino da disciplina: (3) Subgrafos, (4) Grafos Conexos e Componentes, (5) Cortes e Pontes, e (10) Outros tópicos.

Nome:
-------

1. (5,0 pt) [E 1.144] Sejam  $G$  e  $H$  dois grafos conexos tais que  $V_G \cap V_H \neq \emptyset$ . Mostre que o grafo  $G \cup H$  é conexo.
2. (5,0 pt) Na linguagem C, complete as lacunas da função `ehCircuito` conforme apresentada abaixo. Você deve substituir apenas as linhas 6, 12 e 16, pelos trechos de código 1, 2 e 3, respectivamente. O objetivo é que esta função retorne valor 1, se o grafo fornecido como parâmetro for um circuito. O valor de retorno será 0, caso não for.

```
1 int ehCircuito(Grafo *g){
2
3     if( g->n < 3)
4         return 0;
5     if( g->n == 3)
6         // TRECHO 1
7     else{
8         int vGrau2 = -1;
9         int qtdGrau2 = 0;
10        for(int v=0; v<g->n; v++){
11            if( grau(g,v) == 2){
12                // TRECHO 2
13            }
14        }
15
16        //TRECHO 3
17
18        Grafo *h = gMenosV(g, vGrau2);
19        return ehCaminho(h);
20    }
21 }
```

# 1 grafos.h

```
1  #ifndef GRAFO_H_INCLUDED
2  #define GRAFO_H_INCLUDED
3
4  #include <stdlib.h>
5
6  typedef struct grafo {
7      char *nome;
8      int n;          // Numero de vertices
9      int m;          // Numero de arestas
10     int **adj;       // Ponteiro para a matriz de adjacencias
11 } Grafo;
12
13 int **gerarMatriz(int n) {
14
15     int i, j;
16
17     // Aloca um vetor de n ponteiros (do tipo int) na
18     // memoria
19     int **m = malloc( n * sizeof (int *));
20
21     // Para cada ponteiro, aloca n elementos (do tipo int
22     // ) na memoria
23     for (i = 0; i < n; i++)
24         m[i] = malloc( n * sizeof (int));
25
26     // Preenche cada celula da matriz com valor 0
27     for (i = 0; i < n; i++)
28         for (j = 0; j < n; j++)
29             m[i][j] = 0;
30
31     return m;
32 }
33
34 void inicializar(Grafo *g, char *nome, int n) {
35
36     // Inicializa os parametros da estrutura Grafo
37     g->nome = nome;
38     g->n = n;
39     g->m = 0;
40     g->adj = gerarMatriz(n);
41 }
```

```

40 }
41
42 void inserirAresta(Grafo *g, int v, int w) {
43
44     // Garantindo que os argumentos estao entre
45     // 0 e n-1 (fechado)
46     if( (v >= 0) && (v < g->n) ){
47         if( (w >= 0) && (w < g->n) ){
48
49             if (g->adj[v][w] == 0) { // Verifica se a
50                 aresta nao existe
51                 // E necessario preencher as duas celulas
52                 // com 1
53                 g->adj[v][w] = 1;
54                 g->adj[w][v] = 1;
55                 g->m++;
56             }
57         }
58     }
59
60 void removerAresta(Grafo *g, int v, int w) {
61
62     // Garantindo que os argumentos estao entre
63     // 0 e n-1 (fechado)
64     if( (v >= 0) && (v < g->n) ){
65         if( (w >= 0) && (w < g->n) ){
66
67             if (g->adj[v][w] == 1) { // Verifica se a
68                 aresta existe
69                 // Eh necessario preencher as duas
70                 // celulas com 0
71                 g->adj[v][w] = 0;
72                 g->adj[w][v] = 0;
73                 g->m--;
74             }
75         }
76     }
77
78 }

```

```

79 void exibir(Grafo *g) {
80
81     int v, w;
82
83     printf("\n=====");
84     printf("\n%s", g->nome);
85     printf("\n=====");
86
87     // Imprime o conjunto de vertices
88     printf("\nV = {");
89
90     for(v = 0; v < g->n; v++ )
91         printf("%d, ", v);
92
93     if(g->n > 0)
94         printf("\b\b");
95     printf("}");
96
97
98     // Imprime o conjunto de arestas
99     printf("\nA = {");
100     for (v = 0; v < g->n; v++)
101         for (w = 0; w < g->n; w++)
102             if (g->adj[v][w] == 1)
103                 if (v < w)
104                     printf("(%d, %d); ", v, w);
105
106     if(g->m > 0)
107         printf("\b\b");
108     printf("}");
109
110     printf("\n=====\\n");
111 }
112
113 int grau(Grafo *g, int v){
114
115     int soma;
116
117     if( (v >= 0) && (v < g->n) ){
118         soma = 0;
119         for(int i=0; i<g->n; i++)
120             soma += g->adj[v][i];
121     }

```

```

122
123     return soma;
124 }
125
126 int grauMinimo(Grafo *g){
127
128     int minimo = g->n;
129
130     for(int i=0; i < g->n; i++){
131
132         int grauI = grau(g, i);
133
134         if( grauI < minimo )
135             minimo = grauI;
136     }
137
138     return minimo;
139 }
140
141 int grauMaximo(Grafo *g){
142
143     int maximo = -1;
144
145     for(int i=0; i < g->n; i++){
146
147         int grauI = grau(g, i);
148
149         if( grauI > maximo )
150             maximo = grauI;
151     }
152
153     return maximo;
154 }
155
156 int ehRegular(Grafo *g){
157
158     if(g->n != 1){
159         int valorGrau = grau(g, 0);
160
161         for(int i=1; i<g->n; i++)
162             if( grau(g,i) != valorGrau )
163                 return 0;
164     }

```

```

165
166     return 1;
167 }
168
169 int eh3Regular(Grafo *g){
170
171     if(g->n != 1){
172         int valorGrau = grau(g, 0);
173
174         if(valorGrau != 3)
175             return 0;
176
177         for(int i=1; i<g->n; i++)
178             if( grau(g,i) != valorGrau )
179                 return 0;
180
181         return 1;
182     }
183
184     return 0;
185 }
186
187 Grafo *gMenosV(Grafo *g, int v){
188
189     Grafo *h = malloc(sizeof (Grafo));
190     inicializar(h, "G - v", g->n - 1);
191
192     for(int i=0; i<g->n; i++){
193
194         int l = i;
195
196         if(i == v) continue;
197         else
198             if(i > v)
199                 l = i - 1;
200
201         for(int j=0; j<g->n; j++){
202
203             int c = j;
204
205             if(j == v) continue;
206             else
207                 if(j > v)

```

```

208         c = j - 1;
209
210         h->adj[l][c] = g->adj[i][j];
211     }
212 }
213
214     return h;
215 }
216
217 int ehCaminho(Grafo *g){
218
219     if( g->n == 1)
220         return 1;
221     else{
222         //Encontrar um vertice de grau 1 e
223         //contar a qtd de vertices de grau 1.
224         int vGrau1 = -1;
225         int qtdGrau1 = 0;
226         for(int v=0; v<g->n; v++){
227             if( grau(g,v) == 1){
228                 vGrau1 = v;
229                 qtdGrau1++;
230             }
231         }
232
233         //Se nao houver vertice de grau 1 ou
234         //se nao houver dois vertices de grau 1, nao eh
235         caminho
236         if( vGrau1 == -1 || qtdGrau1 != 2)
237             return 0;
238
239         //Criar H = G - v (vertice de grau 1)
240         Grafo *h = gMenosV(g, vGrau1);
241
242         //Chamada recursiva para H
243         return ehCaminho(h);
244     }
245 }
246 #endif // GRAFO_H_INCLUDED

```



## 2 fabrica.h

```
1  #ifndef FABRICA_H_INCLUDED
2  #define FABRICA_H_INCLUDED
3
4  #include <stdlib.h>
5  #include "grafo.h"
6
7  Grafo *k(int n){
8
9      Grafo *g = malloc(sizeof (Grafo));
10
11      inicializar(g, "K_x", n);
12
13      for(int i=0; i<n; i++)
14          for(int j=0; j<n; j++)
15              if(i != j)
16                  inserirAresta(g, i, j);
17
18      return g;
19 }
20
21 Grafo *caminho(int n){ // Comprimento n-1
22
23     Grafo *g = malloc(sizeof (Grafo));
24
25     inicializar(g, "Caminho (Comp. x)", n);
26
27     for(int i=0; i < n-1; i++)
28         inserirAresta(g, i, i+1);
29
30     return g;
31 }
32
33 Grafo *circuito(int n){ // Comprimento n-1
34
35     Grafo *g = caminho(n);
36     inserirAresta(g, 0, n-1);
37     return g;
38 }
39
40 #endif // FABRICA_H_INCLUDED
```