

Questão 1

Não respondido

Vale 20,00  
ponto(s).

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define true 1
4  #define false 0
5  #define INT_MAX 32000
6  typedef int bool;
7  typedef int TIPOPESO;
8
9  typedef struct adjacencia{
10     int vertice;
11     TIPOPESO peso;
12     struct adjacencia *prox;
13 } ADJACENCIA;
14
15 typedef struct vertice{
16     int grau;
17     ADJACENCIA *cab;
18 } VERTICE;
19
20 typedef struct grafo {
21     int vertices;
22     int arestas;
23     VERTICE *adj;
24 } GRAFO;
25
26
27 GRAFO *criarGrafo(int v){
28     GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));
29
30     g->vertices    = v;
31     g->arestas     = 0;
32     g->adj         = (VERTICE *) malloc(v*sizeof(VERTICE));
33     int i;
34
35     for (i=0; i<v; i++){
36         g->adj[i].cab = NULL;
37         g->adj[i].grau=0;
38
39     return g;
40 }
41
42 ADJACENCIA *criaAdj(int v,int peso){
43     ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
44     temp->vertice    = v;

```

```

45     temp->peso      = peso;
46     temp->prox      = NULL;
47     return (temp);
48 }
49
50 bool criaAresta(GRAFO *gr, int vi, int vf, TIPOPESO p){
51     if (!gr)
52         return(false);
53     if((vf<0) || (vf >= gr->vertices))
54         return(false);
55     if((vi<0) || (vf >= gr->vertices))
56         return(false);
57
58     ADJACENCIA *novo = criaAdj(vf,p);
59
60     novo->prox      = gr->adj[vi].cab;
61     gr->adj[vi].cab = novo;
62     gr->adj[vi].grau++;
63
64     ADJACENCIA *novo2 = criaAdj(vi,p);
65
66     novo2->prox      = gr->adj[vf].cab;
67     gr->adj[vf].cab = novo2;
68     gr->adj[vf].grau++;
69
70     gr->arestas++;
71     return (true);
72 }
73
74 void imprime(GRAFO *gr){
75     printf("Vertices: %d. Arestas: %d, \n", gr->vertices,gr->arestas);
76
77     int i;
78     for(i=0;i<gr->vertices; i++){
79         printf("v%d: ",i);
80         ADJACENCIA *ad = gr->adj[i].cab;
81         while(ad){
82             printf("v%d(%d) ", ad->vertice,ad->peso);
83             ad = ad->prox;
84         }
85
86         printf("\n");
87     }
88 }
89
90 void myFunction(GRAFO *gr, int orig, int *pai);

```

```

91
92 int main(){
93
94     GRAFO *gr = criarGrafo(6);
95     criaAresta(gr,0,1,6);
96     criaAresta(gr,0,2,1);
97     criaAresta(gr,0,3,5);
98     criaAresta(gr,1,2,2);
99     criaAresta(gr,1,4,5);
100    criaAresta(gr,2,3,2);
101    criaAresta(gr,2,4,6);
102    criaAresta(gr,2,5,4);
103    criaAresta(gr,3,5,4);
104    criaAresta(gr,4,5,3);
105
106
107    imprime(gr);
108
109    int i, pai[gr->vertices];
110    myFunction(gr,0,pai);
111    for(i=0; i<gr->vertices; i++)
112        printf("%d: %d\n",pai[i],i);
113
114    return 0;
115 }

```

```

void myFunction(GRAFO *gr, int orig, int *pai){
    int i,j,dest,primeiro;
    double menorPeso;

    int *arv = (int *) malloc (gr->vertices * sizeof(int));

    for(i=0; i < gr->vertices; i++){
        arv[i] = i;
        pai[i] = -1;
    }
    pai[orig] = orig;
    while(true){
        primeiro = true;
        for(i=0; i< gr->vertices; i++){
            ADJACENCIA *ad = gr->adj[i].cab;
            while(ad){
                if(arv[i] != arv[ad->vertice]){
                    if(primeiro){
                        menorPeso = ad->peso;
                    }
                }
                ad = ad->prox;
            }
        }
        if(menorPeso != 0){
            j = arv[ad->vertice];
            pai[j] = i;
            arv[j] = arv[i];
        }
        else{
            break;
        }
    }
}

```

```

        menorPeso = ad->peso;
        orig      = i;
        dest      = ad->vertice;
        primeiro = false;
    }else{
        if(menorPeso > ad->peso){
            menorPeso = ad->peso;
            orig = i;
            dest = ad->vertice;
        }
    }
    ad = ad->prox;
}
}
if(primeiro == true) break;
if(pai[orig] == -1 ) pai[orig] = dest;
else pai[dest] = orig;

for(i=0; i < gr->vertices; i++)
    if (arv[i] == arv[dest])
        arv[i] = arv[orig];
}
}

```

Determine:

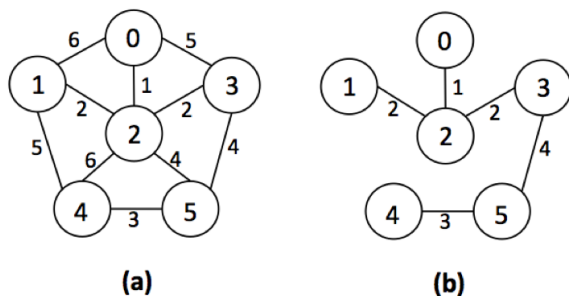
1. Qual algoritmo em grafo a função myFunction executa.
2. Quais as impressões feitas no laço da linha 111 e 112? Explique o que isso significa dentro do problema que o algoritmo da função myFunction busca resolver.
3. Envie para o telegram do professor uma imagem do grafo de entrada e o impresso.

Questão 2

Incorreto

Atingiu 0,00 de 10,00

A Figura (a) abaixo mostra o exemplo de um grafo não direcionado  $G$  com os pesos mostrados ao lado de cada aresta. Sobre a árvore  $T$  representada na Figura (b), é correto afirmar que:



Escolha uma opção:

- ☐ T representa a árvore de caminhos mais curtos do grafo da Figura (a) com origem única no vértice 2. T não é única, pois a substituição da aresta (3,5) pela aresta (2,4) produz caminhos mais curtos entre todos os pares de vértices do grafo.
- ☐ T representa a árvore geradora mínima do grafo da Figura (a) cujo peso total é 12. T não é única, pois a substituição da aresta (3,5) pela aresta (2,5) produz outra árvore geradora de custo 12.
- ☐ T representa a ordenação topológica do grafo da Figura (a). O peso da aresta (0,2) indica que ela deve ser executada antes da aresta (2,3) e o peso da aresta (2,3) indica que ela deve ser executada antes da aresta (4,5) e assim sucessivamente.
- ☒ T representa a árvore de caminhos mais curtos entre todos os pares de vértices do grafo da Figura (a). T não é única, pois a substituição da aresta (3,5) pela aresta (2,5) produz caminhos mais curtos entre os mesmos pares de vértices do grafo. ✗
- ☐ T representa a árvore geradora mínima do grafo da Figura (a) cujo peso total é 12. A substituição da aresta (3,5) pela aresta (2,4) produz uma árvore geradora máxima cujo peso total é 14.

Sua resposta está incorreta.

A resposta correta é: T representa a árvore geradora mínima do grafo da Figura (a) cujo peso total é 12. T não é única, pois a substituição da aresta (3,5) pela aresta (2,5) produz outra árvore geradora de custo 12.

Questão 3

Correto

Atingiu 20,00 de 20,00

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define true 1
4  #define false 0
5  #define INT_MAX 32000
6  typedef int bool;
7  typedef int TIPOPESO;
8
9  typedef struct adjacencia{
10     int vertice;
11     TIPOPESO peso;
12     struct adjacencia *prox;
13 } ADJACENCIA;
14
15 typedef struct vertice{
16     /* Dados armazenados vao aqui */
17     ADJACENCIA *cab;
18 } VERTICE;
19
20 typedef struct grafo {
21     int vertices;
22     int arestas;
23     VERTICE *adj;
24 } GRAFO;
25
26 /* Criando um grafo */
27 GRAFO *criarGrafo(int v){
28     GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));
29
30     g->vertices    = v;
31     g->arestas     = 0;
32     g->adj         = (VERTICE *) malloc(v*sizeof(VERTICE));
33     int i;
34
35     for (i=0; i<v; i++)
36         g->adj[i].cab = NULL;
37
38     return g;
39 }
40
41 ADJACENCIA *criaAdj(int v,int peso){
42     ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
43     temp->vertice    = v;
44     temp->peso       = peso;

```

```

45     temp->prox      = NULL;
46     return (temp);
47 }
48
49 bool criaAresta(GRAFO *gr, int vi, int vf, TIPOPESO p){
50     if (!gr)
51         return(false);
52     if((vf<0) || (vf >= gr->vertices))
53         return(false);
54     if((vi<0) || (vf >= gr->vertices))
55         return(false);
56
57     ADJACENCIA *novo = criaAdj(vf,p);
58
59     novo->prox      = gr->adj[vi].cab;
60     gr->adj[vi].cab = novo;
61
62     ADJACENCIA *novo2 = criaAdj(vi,p);
63
64     novo2->prox      = gr->adj[vf].cab;
65     gr->adj[vf].cab = novo2;
66
67     gr->arestas++;
68     return (true);
69 }
70
71 void imprime(GRAFO *gr){
72     printf("Vertices: %d. Arestas: %d, \n", gr->vertices,gr->arestas);
73
74     int i;
75     for(i=0;i<gr->vertices; i++){
76         printf("v%d: ",i);
77         ADJACENCIA *ad = gr->adj[i].cab;
78         while(ad){
79             printf("v%d(%d) ", ad->vertice,ad->peso);
80             ad = ad->prox;
81         }
82
83         printf("\n");
84     }
85 }
86 void inicializaD(GRAFO *g, int *d, int *p, int s);
87 void relaxa(GRAFO *g, int *d, int *p, int u, int v);
88 bool existeAberto(GRAFO *g, int *aberto);
89 int menorDist(GRAFO *g, int *aberto, int *d);
90 int *dijkstra(GRAFO *g, int s);

```

```

91
92
93
94 int main(){
95
96     GRAFO *gr = criarGrafo(6);
97     criaAresta(gr,0,1,10);
98     criaAresta(gr,0,2,5);
99     criaAresta(gr,2,1,3);
100    criaAresta(gr,1,3,1);
101    criaAresta(gr,2,3,8);
102    criaAresta(gr,2,4,2);
103    criaAresta(gr,4,5,6);
104    criaAresta(gr,3,5,4);
105    criaAresta(gr,3,4,4);
106
107    imprime(gr);
108
109    int *r = dijkstra(gr,0);
110
111    int i;
112    for(i=0; i < gr->vertices; i++)
113        printf("D(v0 -> v%d) = %d\n", i,r[i]);
114    return 0;
115 }
116
117 void inicializaD(GRAFO *g, int *d, int *p, int s){
118     int v;
119     for(v=0; v < g->vertices; v++){
120         d[v] = INT_MAX/2;
121         p[v] = -1;
122     }
123
124     d[s] = 0;
125 }
126
127 void relaxa(GRAFO *g, int *d, int *p, int u, int v){
128     ADJACENCIA *ad = g->adj[u].cab;
129     while (ad && ad->vertice != v)
130         ad = ad->prox;
131     inicializaD(g,d,p,s);
132     if (ad){
133         if ( d[v] > d[u] + ad->peso){
134             d[v] = d[u] + ad->peso;
135             p[v] = u;
136         }
137     }

```



```

137     }
138     } existeAberto(g,aberto)
139     bool existeAberto(GRAFO *g, int *aberto){
140         int i;
141         for(i=0; i < g->vertices; i++)
142             if (aberto[i]) return (true);
143         return(false);
144     }
145     int menorDist(GRAFO *g, int *aberto, int *d){
146         int i;
147         for(i=0; i < g->vertices; i++)
148             if(aberto[i]) break;
149
150         if(i==g->vertices) return (-1);
151         int menor = i;
152
153         for(i=menor+1; i < g->vertices; i++)
154             if(aberto[i] && (d[menor] > d[i]))
155                 menor = i;
156         return (menor);
157     }
158     int *dijkstra(GRAFO *g, int s){
159         int *d = (int *) malloc(g->vertices*sizeof(int));
160
161         int p[g->vertices];
162         bool aberto[g->vertices];
163          ✓
164
165         int i;
166         for(i=0; i<g->vertices; i++)
167             aberto[i] = true;
168
169         while (  ✓ ){
170             int u =  ✓;
171             aberto[u] = false;
172
173             ADJACENCIA *ad = g->adj[u].cab;
174             while(ad){
175                  ✓
176                 ad = ad->prox;
177             }
178         }
179     }
180

```

```
    return(d);  
}
```

Determine a ordem de chamada de cada função do algoritmo de Dijkstra.

Sua resposta está correta.

A resposta correta é:

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #define true 1  
4  #define false 0  
5  #define INT_MAX 32000  
6  typedef int bool;  
7  typedef int TIPOPESO;  
8  
9  typedef struct adjacencia{  
10     int vertice;  
11     TIPOPESO peso;  
12     struct adjacencia *prox;  
13 } ADJACENCIA;  
14  
15 typedef struct vertice{  
16     /* Dados armazenados vao aqui */  
17     ADJACENCIA *cab;  
18 } VERTICE;  
19  
20 typedef struct grafo {  
21     int vertices;  
22     int arestas;  
23     VERTICE *adj;  
24 } GRAFO;  
25  
26 /* Criando um grafo */  
27 GRAFO *criarGrafo(int v){  
28     GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));  
29  
30     g->vertices    = v;  
31     g->arestas     = 0;
```

```

32     g->adj          = (VERTICE *) malloc(v*sizeof(VERTICE));
33     int i;
34
35     for (i=0; i<v; i++)
36         g->adj[i].cab = NULL;
37
38     return g;
39 }
40
41 ADJACENCIA *criaAdj(int v,int peso){
42     ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
43     temp->vertice     = v;
44     temp->peso         = peso;
45     temp->prox         = NULL;
46     return (temp);
47 }
48
49 bool criaAresta(GRAFO *gr, int vi, int vf, TIPO PESO p){
50     if (!gr)
51         return(false);
52     if((vf<0) || (vf >= gr->vertices))
53         return(false);
54     if((vi<0) || (vf >= gr->vertices))
55         return(false);
56
57     ADJACENCIA *novo = criaAdj(vf,p);
58
59     novo->prox        = gr->adj[vi].cab;
60     gr->adj[vi].cab = novo;
61
62     ADJACENCIA *novo2 = criaAdj(vi,p);
63
64     novo2->prox        = gr->adj[vf].cab;
65     gr->adj[vf].cab = novo2;
66
67     gr->arestas++;
68     return (true);
69 }
70
71 void imprime(GRAFO *gr){
72     printf("Vertices: %d. Arestas: %d, \n", gr->vertices,gr->arestas);
73
74     int i;
75     for(i=0;i<gr->vertices; i++){
76         printf("v%d: ",i);
77         ADJACENCIA *ad = gr->adj[i].cab;

```

```

78         while(ad){
79             printf("v%d(%d) ", ad->vertice,ad->peso);
80             ad = ad->prox;
81         }
82
83         printf("\n");
84     }
85 }
86 void inicializaD(GRAFO *g, int *d, int *p, int s);
87 void relaxa(GRAFO *g, int *d, int *p, int u, int v);
88 bool existeAberto(GRAFO *g, int *aberto);
89 int menorDist(GRAFO *g, int *aberto, int *d);
90 int *dijkstra(GRAFO *g, int s);
91
92
93
94 int main(){
95
96     GRAFO *gr = criarGrafo(6);
97     criaAresta(gr,0,1,10);
98     criaAresta(gr,0,2,5);
99     criaAresta(gr,2,1,3);
100    criaAresta(gr,1,3,1);
101    criaAresta(gr,2,3,8);
102    criaAresta(gr,2,4,2);
103    criaAresta(gr,4,5,6);
104    criaAresta(gr,3,5,4);
105    criaAresta(gr,3,4,4);
106
107    imprime(gr);
108
109    int *r = dijkstra(gr,0);
110
111    int i;
112    for(i=0; i < gr->vertices; i++)
113        printf("D(v0 -> v%d) = %d\n", i,r[i]);
114    return 0;
115 }
116
117 void inicializaD(GRAFO *g, int *d, int *p, int s){
118     int v;
119     for(v=0; v < g->vertices; v++){
120         d[v] = INT_MAX/2;
121         p[v] = -1;
122     }
123

```

```

124     d[s] = 0;
125 }
126
127 void relaxa(GRAFO *g, int *d, int *p, int u, int v){
128     ADJACENCIA *ad = g->adj[u].cab;
129     while (ad && ad->vertice != v)
130         ad = ad->prox;
131
132     if (ad){
133         if ( d[v] > d[u] + ad->peso){
134             d[v] = d[u] + ad->peso;
135             p[v] = u;
136         }
137     }
138 }
139 bool existeAberto(GRAFO *g, int *aberto){
140     int i;
141     for(i=0; i < g->vertices; i++)
142         if (aberto[i]) return (true);
143     return(false);
144 }
145 int menorDist(GRAFO *g, int *aberto, int *d){
146     int i;
147     for(i=0; i < g->vertices; i++)
148         if(aberto[i]) break;
149
150     if(i==g->vertices) return (-1);
151     int menor = i;
152
153     for(i=menor+1; i < g->vertices; i++)
154         if(aberto[i] && (d[menor] > d[i]))
155             menor = i;
156     return (menor);
157 }
158 int *dijkstra(GRAFO *g, int s){
159     int *d = (int *) malloc(g->vertices*sizeof(int));
160
161     int p[g->vertices];
162     bool aberto[g->vertices];
163     [inicializaD(g,d,p,s);]
164
165     int i;
166     for(i=0; i<g->vertices; i++)
167         aberto[i] = true;
168
169     while ([existeAberto(g,aberto)]){

```

```
170     int u = [menorDist(g,aberto,d)];
171     aberto[u] = false;
172
173     ADJACENCIA *ad = g->adj[u].cab;
174     while(ad){
175         [relaxa(g,d,p,u,ad->vertice);]
176         ad = ad->prox;
177     }
178 }
179 return(d);
180 }
```

Determine a ordem de chamada de cada função do algoritmo de Dijkstra.

Questão 4

Completo

Atingiu 12,00 de 12,00

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define true 1
4  #define false 0
5
6  #define maxV 1024
7  #define BRANCO 0
8  #define CINZA 1
9  #define PRETO 2
10
11 static int dist[maxV], cor[maxV], pred[maxV];
12
13 int *vetor;
14 int inicio,fim;
15
16 typedef int bool;
17 typedef int TIPOPESO;
18
19 typedef struct adjacencia{
20     int vertice;
21     TIPOPESO peso;
22     struct adjacencia *prox;
23 } ADJACENCIA;
24
25 typedef struct vertice{
26
27     ADJACENCIA *cab;
28 }VERTICE;
29
30
31 typedef struct grafo{
32     int vertices;
33     int arestas;
34     VERTICE *adj;
35 }GRAFO;
36
37
38 /* Criando um grafo */
39 GRAFO *criarGrafo(int v){
40     GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));
41
42     g->vertices = v;
43     g->arestas = 0;
44     g->adj      = (VERTICE *) malloc(v*sizeof(VERTICE));

```

```

45
46     for(int i=0; i<v; i++){
47         g->adj[i].cab = NULL;
48     }
49
50     return g;
51 }
52
53 ADJACENCIA *criarAdj(int v, int peso){
54     ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
55     temp->vertice = v;
56     temp->peso = peso;
57     temp->prox = NULL;
58     return temp;
59 }
60
61
62 bool criaAresta(GRAFO *gr, int vi, int vf, TIPOPESO p){
63
64     if(!gr)
65         return(false);
66     if( (vf < 0) || (vf>= gr->vertices))
67         return (false);
68     if( (vi < 0) || (vi>= gr->vertices))
69         return (false);
70
71     ADJACENCIA *novo = criarAdj(vf,p);
72
73     novo->prox = gr->adj[vi].cab;
74     gr->adj[vi].cab = novo;
75
76
77     ADJACENCIA *novo2 = criarAdj(vi,p);
78     novo2->prox = gr->adj[vf].cab;
79     gr->adj[vf].cab = novo2;
80
81     gr->arestas++;
82
83     return true;
84 }
85
86
87 void imprime(GRAFO *gr){
88
89     printf("vertices %d. Arestas: %d \n", gr->vertices, gr->arestas);
90

```



```

91     for(int i=0; i < gr->vertices; i++){
92         printf("v %d", i);
93         ADJACENCIA *ad = gr->adj[i].cab;
94         while(ad){
95             printf(" adj[%d,%d]", ad->vertice, ad->peso);
96             ad = ad->prox;
97         }
98
99         printf("\n");
100     }
101 }
102
103 void init(int maxN){
104     vetor = (int *) malloc(maxN*sizeof(int));
105     inicio = 0;
106     fim = 0;
107 }
108
109
110 int empty(){
111     return inicio == fim;
112 }
113
114 void put(int item){
115     vetor[fim++] = item;
116 }
117
118 int get(){
119     return vetor[inicio++];
120 }
121
122 void funcaox(GRAFO *gr, int raiz);
123
124
125 void imprime2(GRAFO *gr){
126     for(int v=0; v < gr->vertices; v++){
127         printf("(%d,%d)\n", pred[v], v);
128     }
129 }
130 int main(){
131
132     GRAFO *gr = criarGrafo(12);
133
134     criaAresta(gr,0,1,1);
135     criaAresta(gr,0,2,1);
136     criaAresta(gr,1,3,1);

```

```
137     criaAresta(gr,1,6,1);
138     criaAresta(gr,1,7,1);
139     criaAresta(gr,2,3,1);
140     criaAresta(gr,2,4,1);
141     criaAresta(gr,3,4,1);
142     criaAresta(gr,3,8,1);
143     criaAresta(gr,3,9,1);
144     criaAresta(gr,4,9,1);
145     criaAresta(gr,4,8,1);
146     criaAresta(gr,8,9,1);
147     criaAresta(gr,6,7,1);
148     criaAresta(gr,6,10,1);
149     criaAresta(gr,5,11,1);
150     imprime(gr);
151     funcaoX(gr,0);
152     imprime2(gr);
153     return 0;
154 }
```

```

155
156 void funcao(GRAFO *gr, int raiz){
157
158     int u;
159     init(gr->vertices);
160
161     for(int v=0; v < gr->vertices; v++){
162         dist[v] = maxV;
163         cor[v] = BRANCO;
164         pred[v] = -1;
165     }
166
167     dist[raiz] = 0;
168     cor[raiz] = CINZA;
169     pred[raiz] = -1;
170
171     put(raiz);
172
173     while(!empty()){
174
175         u = get();
176         ADJACENCIA *v = gr->adj[u].cab;
177         while(v){
178             if(cor[v->vertice] == BRANCO){
179                 dist[v->vertice] = dist[u]+1;
180                 pred[v->vertice] = u;
181                 cor[v->vertice] = CINZA;
182
183                 put(v->vertice);
184             }
185             v = v->prox;
186         }
187         cor[u] = PRETO;
188     }
189 }
190
191

```

Determine:

1. Qual algoritmo em grafos a função **funcao** executa?
2. A impressão da linha 152?

3. Qual categoria de estrutura de dados (LIFO ou FIFO) o array vetor representa? Justifique sua resposta

1 - Ele executa o algoritmo de BFS.

2 -

(-1,0)

(0,1)

(0,2)

(2,3)

(2,4)

(-1,5)

(1,6)

(1,7)

(4,8)

(4,9)

(6,10)

(-1,11)

3 - O bfs processa os vértices mais próximos primeiro e depois se afasta da origem. Devido a isso precisamos usar uma estrutura de dados que quando consultada forneça o elemento mais antigo, com base na ordem que foi inserido. Uma fila funciona para esses casos, pois o primeiro a entrar é o primeiro a sair (FIFO).

Comentário:

Questão 5

Correto

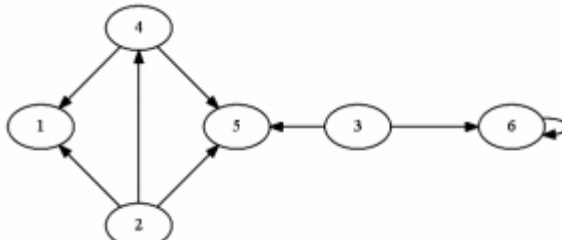
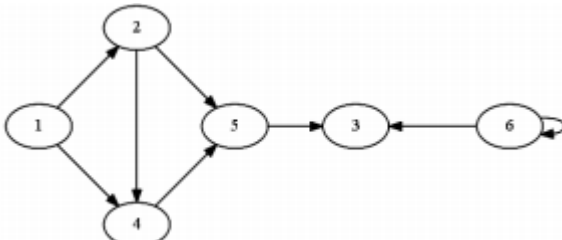
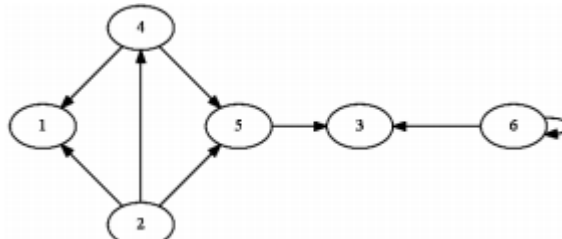
Atingiu 10,00 de 10,00

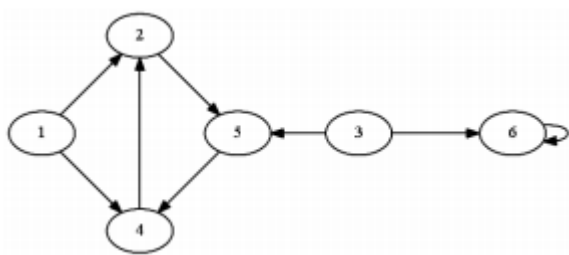
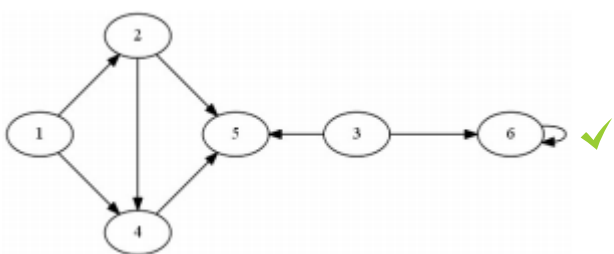
Seja  $G = (V, E)$  um grafo em que  $V$  é o conjunto de vértices e  $E$  é o conjunto de arestas. Considere a representação de  $G$  como uma matriz de adjacências.

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

O correspondente grafo orientado  $G$  é:

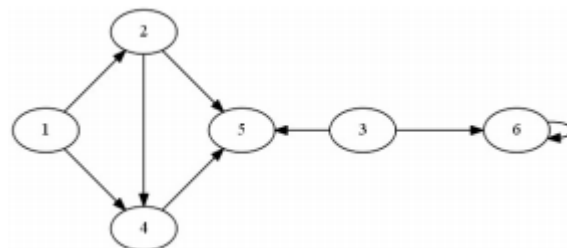
Escolha uma opção:

- ☐ a. 
- ☐ b. 
- ☐ c. 

- ☐ d. 
- ☒ e. 

Sua resposta está correta.

A resposta correta é:




Questão 6

Correto

Atingiu 10,00 de 10,00

Sejam  $G = (V, E)$  um grafo conexo não orientado com pesos distintos nas arestas e  $e \in E$  uma aresta fixa, em que  $|V| = n$  é o número de vértices e  $|E| = m$  é o número de arestas de  $G$ , com  $n \leq m$ . Com relação à geração da árvore de custo mínimo de  $G$ ,  $AGM_G$ , assinale a alternativa correta.

Escolha uma opção:

- ☐ Quando  $e$  está num ciclo em  $G$  e tem o peso da aresta de maior peso neste ciclo então  $e$  garantidamente não estará numa  $AGM_G$ .
- ☐ Quando  $e$  tem o peso da aresta com o maior peso em  $G$  então  $e$  garantidamente não estará numa  $AGM_G$ .
- ☒ Quando  $e$  tem o peso maior ou igual ao da aresta com o  $n$ -ésimo menor peso em  $G$  então  $e$  pode estar numa  $AGM_G$ . 
- ☐ Quando  $e$  tem o peso distinto do peso de qualquer outra aresta em  $G$  então pode existir mais de uma  $AGM_G$ .
- ☐ Quando  $e$  tem o peso da aresta com o  $(n - 1)$ -ésimo menor peso de  $G$  então  $e$  garantidamente estará numa  $AGM_G$ .

Sua resposta está correta.

A resposta correta é: Quando  $e$  tem o peso maior ou igual ao da aresta com o  $n$ -ésimo menor peso em  $G$  então  $e$  pode estar numa  $AGM_G$ .

Universidade Federal de Mato Grosso - UFMT  
Secretaria de Tecnologia da Informação - STI  
Av. Fernando Correa da Costa, nº 2367 - Bairro Boa Esperança. Cuiabá - MT - 78060-900

Fone: +55 (65) 3615-8028

Contato: ces@ufmt.br