Iniciado em	Wednesday, 27 Aug 2025, 06:42
Estado	Finalizada
Concluída em	Wednesday, 27 Aug 2025, 09:54
Tempo empregado	3 horas 11 minutos
Avaliar	Ainda não avaliado

Questão 1

Completo

Vale 1,00 ponto(s).

```c
#include <stdio.h>
#include <stdlib.h>


#define true 1
#define false 0
typedef int bool;
typedef int TIPOPESO;
/* Vértices de grafos são representados por objetos do tipo vertex. */
#define maxV 1024
#define BRANCO 0
#define CINZA 1
#define PRETO 2

static int pa[1000];
static int cnt, d[maxV], f[maxV],dist[maxV],cor[maxV],pred[maxV];
int t = 0;



typedef struct adjacencia{
    int vertice;
    TIPOPESO peso;
    struct adjacencia *prox;
} ADJACENCIA;

typedef struct vertice{
    /* Dados armazenados vao aqui */
    ADJACENCIA *cab;
} VERTICE;

typedef struct grafo {
    int vertices;
    int arestas;
    VERTICE *adj;
} GRAFO;

typedef struct no{
    int u;
    ADJACENCIA *p;
}NO;

NO *vetor;
int fim;
```

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

```
 90
 91 void imprime(GRAFO *gr){
 92     printf("Vertices: %d. Arestas: %d, \n", gr->vertices,gr->arestas);
 93
 94     int i;
 95     for(i=0;i<gr->vertices; i++){
 96         printf("v%d: ",i);
 97         ADJACENCIA *ad = gr->adj[i].cab;
 98         while(ad){
 99             printf("v%d(%d) ", ad->vertice,ad->peso);
100             ad = ad->prox;
101         }
102
103         printf("\n");
104     }
105 }
106
107 void init(int maxN){
108     vetor       = (NO*) malloc (maxN*sizeof(NO));
109     fim  = 0;
110 }
111
112 int empty(){
113     return fim == 0;
114 }
115
116 void put(int item, ADJACENCIA *px){
117     vetor[fim].u = item;
118     vetor[fim].p = px;
119     fim++;
120 }
121
122 NO get(){
123     return vetor[--fim];
124 }
125
126 void free(){
127     free(vetor);
128
129 }
130
131 void funcaox(GRAFO *G, int raiz);
132
133 void imprime2(GRAFO *gr){
134     for(int v=0; v < gr->vertices; v++){
```

```
        printf("(%d,%d)\n",pred[v],v);
    }

}

int main(){

    GRAFO *gr = criarGrafo(12);
    criaAresta(gr,0,1,1);
    criaAresta(gr,0,2,1);
    criaAresta(gr,1,3,1);
    criaAresta(gr,1,6,1);
    criaAresta(gr,1,7,1);
    criaAresta(gr,2,3,1);
    criaAresta(gr,2,4,1);
    criaAresta(gr,3,4,1);
    criaAresta(gr,3,8,1);
    criaAresta(gr,3,9,1);
    criaAresta(gr,4,9,1);
    criaAresta(gr,4,8,1);
    criaAresta(gr,8,9,1);
    criaAresta(gr,6,7,1);
    criaAresta(gr,6,10,1);
    criaAresta(gr,5,11,1);

    imprime(gr);
    funcaox(gr,0);
    imprime2(gr);

    return 0;
}
```

Determine:

1. Qual algoritmo em grafos a função **funçãox** executa?
2. a impressão da linha 162?
3. Qual categoria de estrutura de dados (LIFO ou FIFO) o array vetor representa? Justifique sua resposta

1) Busca em profundidade

2) (null,0)
|0| ->1,2
|1| -> 3,6,7
|2| -> 3,4
|3| -> 4,8,9
|4| -> 9,8
|5| -> 11
|6| -> 7,10
|8| -> 9

3) Tipo LIFO (pilha)

Questão **2**

Incorreto

Atingiu 0,00 de 1,00

Dado o seguinte grafo

fasd

O grafo `G` tem conjunto de arestas E={`(0,2),(0,5),(0,7),(1,7),(1,3),(1,5),(2,6),(3,4),(3,5),(4,6),(4,7)`} e portanto suas listas de adjacências são

```
0: 2 5 7
1: 3 5 7
2: 0 6
3: 1 4 5
4: 3 6 7
5: 0 3 1
6: 2 4
7: 0 1 4
```

Sendo a origem em 0, determine o conjunto de arestas E'=(pred[v],v) para um busca em largura.

Obs. O conjunto das arestas deve estar ordenado pelo v, ou seja, a resposta dever ser: {(-1,0),(y,1),(x,2)....}

Resposta: ✖

A resposta correta é: {(-1,0),(5,1),(0,2),(5,3),(7,4),(0,5),(2,6),(0,7)}

Questão 3

Incorreto

Atingiu 0,00 de 1,00

Dado o seguinte grafo

g1

O grafo G tem conjunto de arestas E={(0,2),(0,5),(0,7),(1,7),(2,6),(3,4),(3,5),(4,5),(4,6),(4,7)} e, portanto, suas lista de adjacências são

```
0: 2 5 7
1: 7
2: 0 6
3: 4 5
4: 3 5 6 7
5: 0 3 4
6: 2 4
7: 0 1 4
```

Sendo a origem em 0, determine o conjunto de arestas E'=(pred[v],v) para um busca em profundidade.

Obs. O conjunto das arestas deve estar ordenado pelo v, ou seja, a resposta dever ser: {(-1,0),(y,1),(x,2)....}

Resposta: ✖

A resposta correta é: {(-1,0),(7,1),(0,2),(4,3),(6,4),(3,5),(2,6),(4,7)}

Questão **4**

Incorreto

Atingiu 0,00 de 1,00

Dado o seguinte grafo

fasdfs

Determine o conjunto de arestas E', que representa a Árvore Geradora Mínima para o grafo acima utilizando o algoritmo de Kruskal.

Obs. O conjunto das arestas deve estar ordenado pelo v, ou seja, a resposta dever ser: {(0,0),(y,1),(x,2)....}

Resposta: ✖

A resposta correta é: {(0,0),(4,1),(0,2),(0,3),(3,4),(3,5),(4,6),(4,7),(9,8),(6,9)}

Questão **5**

Correto

Atingiu 1,00 de 1,00

```c
#include <stdio.h>
#include <stdlib.h>
#define true 1
#define false 0
#define INT_MAX 32000
typedef int bool;
typedef int TIPOPESO;

typedef struct adjacencia{
    int vertice;
    TIPOPESO peso;
    struct adjacencia *prox;
} ADJACENCIA;

typedef struct vertice{
    /* Dados armazenados vao aqui */
    ADJACENCIA *cab;
} VERTICE;

typedef struct grafo {
    int vertices;
    int arestas;
    VERTICE *adj;
} GRAFO;

/* Criando um grafo */
GRAFO *criarGrafo(int v){
    GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));

    g->vertices     = v;
    g->arestas      = 0;
    g->adj          = (VERTICE *) malloc(v*sizeof(VERTICE));
    int i;

    for (i=0; i<v; i++)
        g->adj[i].cab = NULL;

    return g;
}

ADJACENCIA *criaAdj(int v,int peso){
    ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
    temp->vertice   = v;
    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
132     if (ad){
133         if ( d[v] > d[u] + ad->peso){
134             d[v] = d[u] + ad->peso;
```

```
            p[v] = u;
        }
    }
}
bool existeAberto(GRAFO *g, int *aberto){
    int i;
    for(i=0; i < g->vertices; i++)
        if (aberto[i]) return (true);
    return(false);
}
int menorDist(GRAFO *g, int *aberto, int *d){
        int i;
        for(i=0; i < g->vertices; i++)
            if(aberto[i]) break;

        if(i==g->vertices) return (-1);
        int menor = i;

        for(i=menor+1; i < g->vertices; i++)
            if(aberto[i] && (d[menor] > d[i]))
                menor = i;
        return (menor);
}
int *dijkstra(GRAFO *g, int s){
    int *d = (int *) malloc(g->vertices*sizeof(int));

    int p[g->vertices];
    bool aberto[g->vertices];
    inicializaD(g,d,p,s);

    int i;
    for(i=0; i<g->vertices; i++)
        aberto[i] = true;

    while (existeAberto(g,aberto)){

        int u = menorDist(g,aberto,d);
        aberto[u] = false;

        ADJACENCIA *ad = g->adj[u].cab;
        while(ad){
            relaxa(g,d,p,u,ad->vertice);

```

```
            ad = ad->prox;
        }
180     }
    return(d);
}
```

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

Sua resposta está correta.

A resposta correta é:

```c
#include <stdio.h>
#include <stdlib.h>
#define true 1
#define false 0
#define INT_MAX 32000
typedef int bool;
typedef int TIPOPESO;

typedef struct adjacencia{
    int vertice;
    TIPOPESO peso;
    struct adjacencia *prox;
} ADJACENCIA;

typedef struct vertice{
    /* Dados armazenados vao aqui */
    ADJACENCIA *cab;
} VERTICE;

typedef struct grafo {
    int vertices;
    int arestas;
    VERTICE *adj;
} GRAFO;

/* Criando um grafo */
GRAFO *criarGrafo(int v){
    GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));

    g->vertices     = v;
    g->arestas      = 0;
    g->adj          = (VERTICE *) malloc(v*sizeof(VERTICE));
    int i;

    for (i=0; i<v; i++)
        g->adj[i].cab = NULL;

    return g;
}

ADJACENCIA *criaAdj(int v,int peso){
    ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
    temp->vertice   = v;
    temp->peso      = peso;
    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
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 6

Correto

Atingiu 1,00 de 1,00

Em Grafo orientado adjacência não é simétrica.

Determine a validade desta afirmação.

Escolha uma opção:

- Verdadeiro ✔
- Falso

A resposta correta é 'Verdadeiro'.

Questão 7

Correto

Atingiu 1,00 de 1,00

Em Grafo não-orientado a adjacência é não-simétrica.

Determine a validade desta afirmação.

Escolha uma opção:

- Verdadeiro
- Falso ✔

Resposta correta.

A resposta correta é 'Falso'.

Questão **8**

Correto

Atingiu 1,00 de 1,00

Dado um grafo capacitado com fonte em S e término em t, encontrar um fluxo de intensidade máxima entre os respeitam as capacidades dos arcos $(f_{v,w} <= c_{v,w})$.

Selecione o nome do problema que corresponde a esta definição.

- (●) Problema do fluxo máximo ✔
- ( ) Problema do caminho do menor caminho de única origem
- ( ) Problema da Árvore geradora mínima

Sua resposta está correta.

A resposta correta é:
Problema do fluxo máximo

Questão **9**

Incorreto

Atingiu 0,00 de 1,00

Considere a seguinte rede, em que os números nos arcos representam a capacidade do arco (quantidade de fluxo que pode atravessar):

grafo

- Determine o fluxo máximo possível (entre os nós 1 e 7)
- Entre em papel a representação dos fluxos na rede na situação de fluxo máximo;

Resposta: ✖

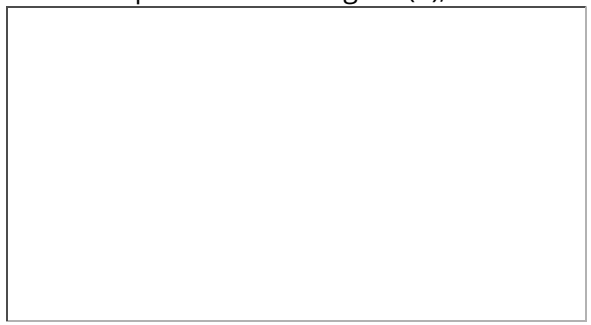
A resposta correta é: 34

Questão **10**

Correto

Atingiu 1,00 de 1,00

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 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 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 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 correta.

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 **11**

Correto

Atingiu 1,00 de 1,00

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 ≤ m. Com relação à geração da árvore de custo mínimo de G, AGM_G, assinale a alternativa correta.

Escolha uma opção:

- 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 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 (n − 1)-ésimo menor peso de G então e garantidamente 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 distinto do peso de qualquer outra aresta em G então pode existir mais de uma 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.