Resolução Alguns Exercícios Lista 1

7 - 0/0 → 0/7 → 5/2 → 0/2 → 2/0 → 2/7 → 5/4

8 - Um grafo quadrado

10 - O número de arestas é igual à metade da soma dos graus, portanto a resposta é 7.

11 - Grafo não orientado: 2m; orientado: m.

12 - 3v = 70; v = 70/3 = 23,33; portanto a resposta é 23.

13 - n (n-1) = 2m; m = (n(n-1))/2; m = $(n^2-n)/2$ ← resposta 1.
n (n-1) = m; m = n^2-n ← resposta 2

15 - 5

16 - Usar exemplo

17 - a) u, v4, v5, w; b) tá errado, tem que ser caminho

18 -Não (todos vértices precisariam ter grau par) e não (tem que mostrar na mão)

19 - É um grafo circular, conexo e cada vértice tem grau 2

20 - a) (questão complicada sem números nos vértices) 2,1; 2,5; 3,1; 3,5; 5,1
b) 2,1; 2,5; 3,1; 3,5; 5,1
c) 2,1; 2,5; 3,1; 3,5
d) 1,...; 2,3; 2,4; 3,4; 3,2; 4, …; 5,2; 5,3; 5,4

31 - A linearização pode ser feita de várias formas diferentes, por ex.: a) alocando-se uma quantidade pré-definida de memória para cada vértice; b) inserindo uma informação extra na lista com a contagem de arestas de cada vértice; ou c) com uma posição especial na lista de cada vértice indicando o final da lista. A linearização é útil quando os vértices e arestas do grafo são relativamente estáticos, ou seja, não se modificam (inseridos, removidos, etc.) regularmente. A linearização é útil quando o número de arestas de cada vértice é pré-definida, como em uma árvore binária ou um grafo regular. Nesse caso, é possível alocar préviamente a memória do armazenamento do grafo.

38 - Função lista_para_matriz {
```
    Matriz.init 0
        para_cada v1 em Lista {
            aresta = v1.prox
            enquanto aresta != nulo {
                Matriz[v1.id][aresta.alvo] = 1
                aresta = aresta.prox
            }
        }
}
```

39 - Função matriz_para_lista {
```
Lista.init N
para i = 1 .. N {
    v1 = Lista[i]
    para j = i+1 .. N {
        se Matriz[i][j] == 1 {
            v2 = Vertice.novo j
            v1.prox = v2
            v1 = v2
        }
    }
}
}
```

40 - a) Matriz de Adj.: O(|V^2|)
b) Lista de Ponteiros: O(|V| + |A|)
c) Lista de Ponteiros: O(|V| + 2|A|)

41 - a) Função remover_vertices_isolados {
```
Marcados = Boolean[].novo N, true
para i = 1 .. N {
    aresta = Lista[i].prox
    se aresta != nulo {
        Marcados[i] = false
    }
    enquanto aresta != nulo {
        Marcados[aresta.alvo] = false
        aresta = aresta.prox
    }
}
para i = 1 .. N {
    se Marcados[i] == true {
        Lista.remover i
    }
}
}
```

b)Função grafo_é_conexo? {
```
// igual à anterior, mudando apenas o final:
para i = 1 .. N {
    se Marcados[i] == true {
        retorne false
    }
}
retorne true
}
```

c)Função eliminar_ciclos (vertice, pred) {
```
vertice.marcar
aresta = vertice.prox
enquanto aresta != nulo e aresta.alvo != pred {
    prox = aresta.prox
    se aresta.alvo.é_marcado? {
```

```
                vertice.remover aresta
            } senão {
                eliminar_ciclos aresta.alvo, vertice
            }
            aresta = prox
        }
        vertice.desmarcar
    }
```