Algoritmos e Estruturas de Dados II

Exercícios – Revisão para P2

1) (Questão ENADE 2011)

Considere que G é um grafo qualquer e que V e E são os conjuntos de vértices e de arestas de G, respectivamente. Considere também que grau(v) é o grau de um vértice v pertencente ao conjunto V. Nesse contexto, analise as seguintes asserções.

Em G, a quantidade de vértices com grau ímpar é ímpar.

PORQUE

Para G, vale a identidade dada pela expressão

$$\sum_{v \in V} grau(v) = 2|E|$$

Acerca dessas asserções, assinale a opção correta.

A. As duas asserções são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
B. As duas asserções são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira.
C. A primeira asserção é uma proposição verdadeira, e a segunda uma proposição falsa.
D. A primeira asserção é uma proposição falsa, e a segunda uma proposição verdadeira.
E. Tanto a primeira quanto a segunda asserções são proposições falsas.

2) A partir da matriz de adjacência e lista de vértices representados abaixo, responda as seguintes questões.

	0	1	2	3	4	5
0	0	40	0	20	0	0
1	0	0	10	0	0	25
2	10	0	0	35	0	0
3	0	0	0	0	15	0
4	0	0	0	0	0	10
5	0	0	15	0	0	0

0	1	2	3	4	5
G	R	A	F	O	S

a) Desenhe uma representação gráfica deste grafo. Procure desenhar um grafo planar (sem cruzamento de arestas).

b) Faça os caminhamentos em largura e profundidade iniciando pelo vértice A. Utilize o menor valor da aresta para critério de visita entre os vértices adjacentes.

c) É possível definir uma ordem topológica sobre o grafo? Se sim, apresente uma possível ordem topológica. Se não, justifique.

d) Redesenhe o grafo da questão “a” desconsiderando a orientação das arestas. Apresente a árvore geradora de custo mínimo através de um dos dois algoritmos estudados em aula. Informe o nome do algoritmo utilizado.

e) Considerando o grafo da questão “a”, quais são os caminhos mínimos a partir do vértice G para todos os demais vértices. Faça a análise através do algoritmo de Dijkstra. Mostre a execução passo a passo indicando:

- a evolução dos estados da fila auxiliar durante a busca, a cada iteração do laço;
- o que é visitado (que vértice é escolhido como parte do caminho) em cada iteração do laço;
- o que é marcado em cada iteração do laço.

Iteração (enquanto fila Q não está vazia)	Vértice Origem	Vértice Adjacente Corrente	Fila Prioridade Q	Tabela D[] de valores dos caminhos
1				
2				
...			...	...

3) Qual a complexidade da operação que retorna todos os vértices adjacentes a um determinado vértice, quando o grafo está implementado como uma matriz de adjacências, e os vértices armazenados em uma lista? Explique sua resposta.

4) Considerando um grafo dirigido, representado por matriz de adjacências, construa um algoritmo para verificar se um determinado vértice v é alcançável a partir de um determinado vértice u. Utilize a seguinte assinatura para o algoritmo:

```
boolean éAlcançável(Grafo g, Vertice u, Vertice v)
```

5) Escreva um algoritmo que verifique se um grafo não-dirigido é uma árvore. Dica: um grafo não-dirigido é uma árvore se ele é conexo e não possui ciclos; um grafo não-dirigido é conexo se cada par de vértices u e v possui um caminho entre u e v.

GABARITO:

1) D

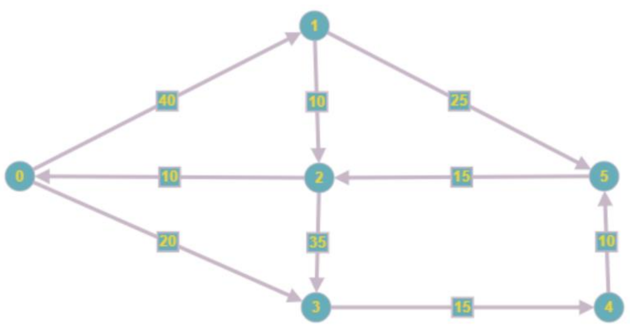
Segundo Cormen et al. (2009), o grau de um vértice em um grafo não direcionado é o número de arestas incidentes. Ainda segundo os mesmos autores, o grau de um vértice em um grafo direcionado é a soma dos graus de entrada e de saída do vértice.

Imaginemos um grafo G (V, E), sendo V = {a, b} e E = {(a, b)}, os dois vértices têm grau ímpar. Nesse caso, temos um grafo que invalida a primeira asserção. Existe apenas uma aresta, então grau(a) = 0 + 1 = 1 e grau(b) = 1 + 0 = 1, no caso de grafos direcionados. No caso de grafos não direcionados, grau(a) = 1 e grau(b) = 1, o que mantém o resultado. A validade da asserção é determinada por um contraexemplo, como é o caso da maior das questões que solicita a avaliação de uma regra universal. Com isso, são eliminadas as alternativas A, B e C.

Toda aresta conecta dois vértices, logo, cada aresta contribui com o grau total do grafo duas vezes, uma em cada vértice que conecta. Portanto, a segunda asserção é verdadeira. Existe um caso particular, em que uma aresta pode conectar um vértice a ele mesmo, formando um arco. Neste caso a contribuição se acumula em um mesmo vértice. Mesmo assim, o arco gera duas contribuições na soma de todos os graus do grafo, o que mantém o resultado.

2)

a)



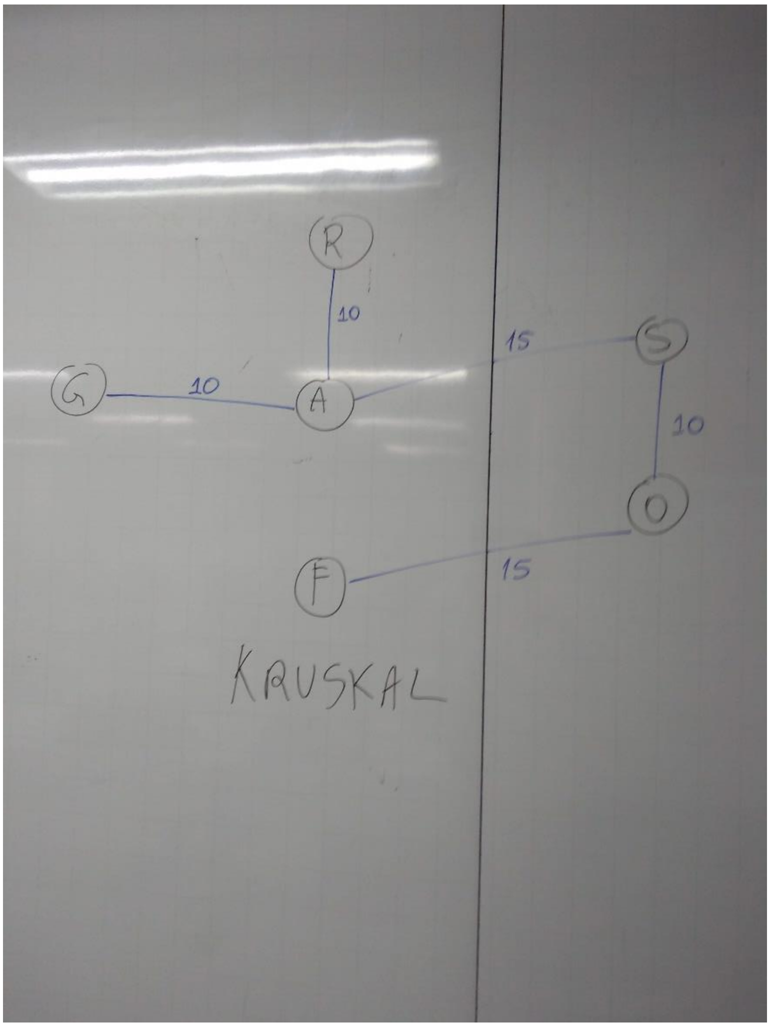
b)

largura: 2, 0, 3, 1, 4, 5

profundidade: 2, 0, 3,4,5,1

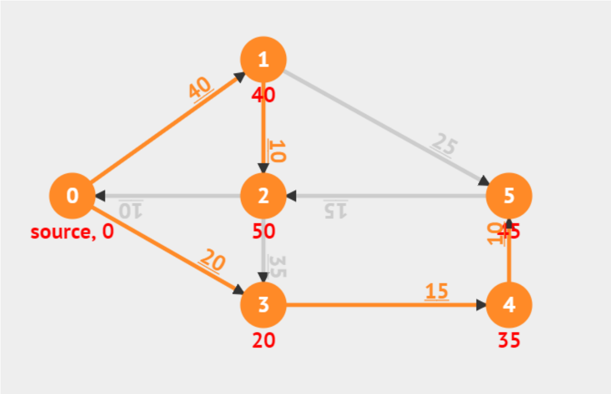
c) Não é possível, pois o grafo possui ciclos. Por exemplos, 0 – 1 – 2-0 é um caminho que define um ciclo.

d)



e)

Iteração (enquanto fila Q não está vazia)	Nodo Origem	Nodo Adjacente Corrente	Fila Prioridade Q	Tabela D[] de valores dos caminhos
0	0		0 1 2 3 4 5	0 ∞ ∞ ∞ ∞ ∞
1	0	1	1 2 3 4 5	0 40 ∞ ∞ ∞ ∞
2	0	3	3 1 2 4 5	0 40 ∞ 20 ∞ ∞
3	3	4	4 1 2 5	0 40 ∞ 20 35 ∞
4	4	5	1 5 2	0 40 ∞ 20 35 45
5	1	2	5 2	0 40 50 20 35 45
6	1	5	5 2	0 40 50 20 35 45
7	5	2	2	0 40 50 20 35 45
8	2			0 40 50 20 35 45



3)

Operação adjacentes(u). Assumir que estamos percorrendo e coletando.

Com matriz de adjacências: O(n), n é o número de vértices do grafo

Com lista de adjacências: O(n), n é o número de arestas de saída

4)

```
boolean éAlcançável(Grafo g, Nodo u, Nodo v) {
 seja marked um array de booleanos de tamano G.V inicializado em falso
 dfs(G,u)
 retorne marked[v]
}

dfs(Grafo G, Nodo v) {
 maked[v] = true;
 para w de 0 a G.V {
  se adj[v][w] != 0 { //w é adjacente a v
    se !marked[w] {
     dfs(G,w)
    }
  }
 }
}
```

5)

//percorrer em BFS ou DFS a partir de qualquer vértice;

//se todos os vértices forem alcançados, é conexo.

//percorrer em DFS;

//para cada vértice v sendo visitado, se existe um

//adjacente u tal que u já foi visitado e u não é pai de v, então existe um ciclo

```
boolean éArvore(Grafo g) {
 seja marked um array de booleanos de tamano G.V inicializado em falso
 //percorre a partir do vértice 0 e verificar se tem ciclo
 se éCiclicoDFS(g,0,-1) {
   retorne falso
 }
 //verificar se alcançou todos os vértices
 para cada vértice v de g {
   se (!marked[v]) {
           retorne falso
   }
 }
 retorne verdadeiro
}

boolean éCiclicoDFS(Grafo g, Nodo v, Nodo pai) {
  marked[v] = verdadeiro
  para cada w adjacente a v {
   se (!marked(w)) {
     se (éCiclicoDFS(g,w,v)) {
           retorne verdadeiro
     }
   } senão {
```

```
    se (w != pai) {
        retorne verdadeiro
    }
   }
  }
  retorne falso;
}
```