2º Teste de Análise e Desenho de Algoritmos

Duração: 2 horas e 15 minutos

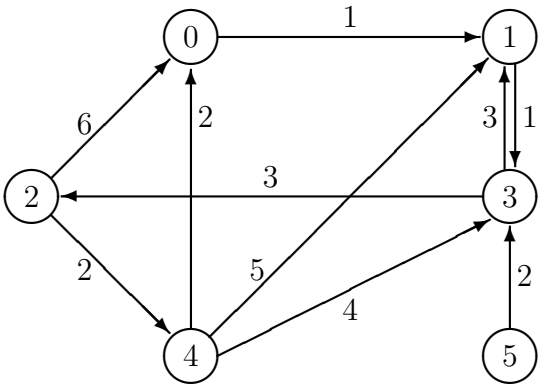
Departamento de Informática, FCT NOVA

3 de Junho de 2017

O teste tem **4** páginas e **4** perguntas

Número: _____ **Nome:** _____

1. Suponha que se executa o algoritmo de Dijkstra com o grafo esquematizado na figura e o vértice origem 2.



(a) [1 valor] Indique a ordem pela qual os vértices são selecionados (retirados da fila com prioridade e passados a *exploreNode* como argumento).

(b) [1 valor] Identifique os arcos cujo tratamento termina numa execução do método *decreaseKey*.

(c) [2 valores] Indique o resultado da função *dijkstra*, ou seja, o conteúdo dos dois vetores retornados.

`length:`

0	1	2	3	4	5

`via:`

0	1	2	3	4	5

2. [6 valores] A classe *WhiteRedGraph* implementa grafos não orientados onde cada arco é branco ou vermelho (mas não pode ter as duas cores).

```
import java.util.List;
import java.util.LinkedList;

public class WhiteRedGraph {

  private boolean[][] edges; // adjacency matrix
  private boolean[][] isRed; // if edge exists: F is white; T is red
  private List<Integer>[] adjNodes; // possible white adjacent nodes

  @SuppressWarnings("unchecked")
  public WhiteRedGraph( int numNodes ) {
      edges = new boolean[numNodes][numNodes];
      isRed = new boolean[numNodes][numNodes];
      adjNodes = new List[numNodes];
      for ( int i = 0; i < numNodes; i++ )
          adjNodes[i] = new LinkedList<Integer>();
  }

  public void addEdge( int node1, int node2 ) {
      if ( node1 != node2 && !edges[node1][node2] ) {
          edges[node1][node2] = true;
          edges[node2][node1] = true;
          // the new edge is white
          isRed[node1][node2] = false; adjNodes[node1].add(node2);
          isRed[node2][node1] = false; adjNodes[node2].add(node1);
      }
  }

  public void removeEdge( int node1, int node2 ) {
      edges[node1][node2] = false;
      edges[node2][node1] = false;
  }

  public boolean isEdgeRed( int node1, int node2 ) {
      return edges[node1][node2] && isRed[node1][node2];
  }

  public void colourIncidentEdges( int node ) {
      for ( int v : adjNodes[node] ) {
          isRed[node][v] = true;
          isRed[v][node] = true;
      }
      adjNodes[node].clear();
  }

}
```

Considere a função $\Phi(G)$, que atribui a cada objeto G da classe *WhiteRedGraph* a soma dos tamanhos das listas ligadas guardadas no vetor G.`adjNodes`:

$$\Phi(G) = \sum_{i=0}^{G.\texttt{adjNodes.length}-1} G.\texttt{adjNodes}[i].\texttt{size()}$$

Prove que Φ é uma função potencial válida e calcule as complexidades amortizadas dos métodos *addEdge*, *removeEdge*, *isEdgeRed* e *colourIncidentEdges*, justificando. Assuma que:

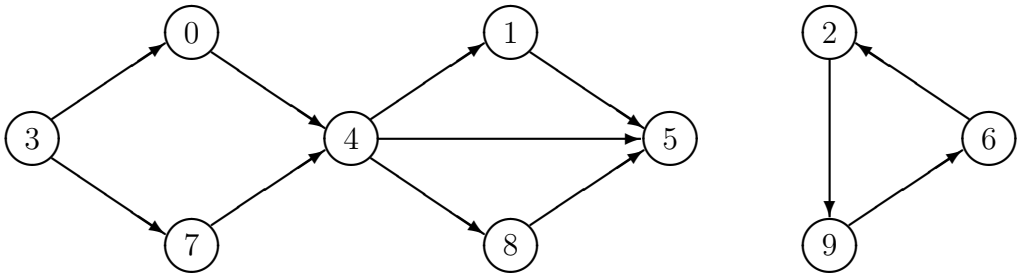
- os argumentos dos métodos são sempre válidos (o valor de `numNodes` é positivo e os valores de `node1`, `node2` e `node` variam entre 0 e G.`adjNodes.length` $- 1$);
- os métodos *add* e *clear* das listas ligadas têm complexidade constante.

No estudo da complexidade amortizada do método *addEdge*, analise separadamente os casos em que: é inserido um novo arco; o conjunto dos arcos não é alterado.

3. [6 valores] Sejam $G = (V, A)$ um grafo orientado e n um inteiro positivo. Uma *coloração dos vértices de G com n cores* é uma função $f : V \longrightarrow \{1, 2, \ldots, n\}$ tal que:

$$\text{para todo o caminho da forma } v_1\, v_2\, v_3 \text{ em } G, \quad |\{\, f(v_1),\, f(v_2),\, f(v_3)\,\}| \;\geq\; 2.$$

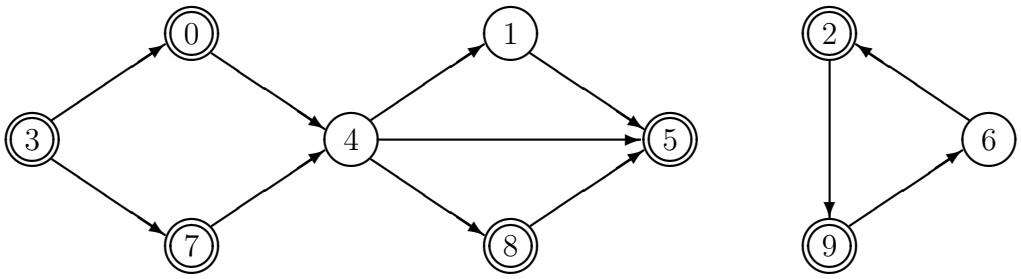
Intuitivamente, f atribui a cada vértice uma das n cores disponíveis de tal forma que todo o caminho de comprimento dois em G passa por, pelo menos, dois vértices de cores distintas.



Por exemplo, a função $\beta : \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \longrightarrow \{1, 2\}$ que atribui:

- 1 aos vértices 1, 4 e 6;
- 2 aos vértices 0, 2, 3, 5, 7, 8 e 9

é uma coloração com duas cores dos vértices do grafo esquematizado nas figuras. A figura de baixo facilita a verificação desta afirmação.



O **Problema da Coloração de Vértices em Grafos Orientados** formula-se da seguinte forma. Dados um grafo orientado $G = (V, A)$ e um inteiro positivo n, existe uma coloração dos vértices de G com n cores?

Prove que o Problema da Coloração de Vértices em Grafos Orientados é NP-completo.

4. [4 valores] A República do Faz-de-Conta vai organizar uma reunião mundial nas instalações onde decorreu o encontro de 1962. Um dos problemas reside nas tomadas dos edifícios. Há tomadas dos tipos que existiam em 1962, mas alguns desses tipos já não são usados e surgiram tipos novos. Nenhum dispositivo pode ser ligado a mais do que uma tomada e nenhuma tomada pode fornecer eletricidade a mais do que um dispositivo.

   Para tentar minimizar o número de dispositivos que não são ligados a uma tomada, vão ser comprados adaptadores. Um adaptador do tipo X-Y permite converter uma tomada do tipo X numa tomada do tipo Y. Há vários tipos de adaptadores, mas não há adaptadores para todos os pares de tipos de tomadas/fichas. É possível adquirir um número tão grande quanto se queira de adaptadores de cada um dos tipos existentes.

   O objetivo é descobrir o número mínimo de dispositivos que não podem ser ligados a uma tomada, conhecendo:

   - os tipos de tomadas/fichas que existem (representados pelos números $0, 1, \ldots, N-1$);
   - a quantidade de tomadas que existem de cada tipo;
   - a quantidade de fichas que existem de cada tipo;
   - os tipos de adaptadores que existem.

Tomadas	
Tipo	Qtd
0	1
1	1
2	1
3	1
4	2
5	0

Fichas	
Tipo	Qtd
0	0
1	3
2	1
3	0
4	0
5	1

Adaptadores	
Ficha	Tomada
0	5
5	1
3	5

   A resposta ao exemplo acima é 1, porque só não é possível ligar um dispositivo do tipo 1:

   - A tomada do tipo 0 recebe um adaptador do tipo 0-5, que recebe um adaptador do tipo 5-1, que recebe um dispositivo com ficha do tipo 1.
   - A tomada do tipo 1 recebe um dispositivo com ficha do tipo 1.
   - A tomada do tipo 2 recebe o dispositivo com ficha do tipo 2.
   - A tomada do tipo 3 recebe um adaptador do tipo 3-5, que recebe o dispositivo com ficha do tipo 5.
   - As tomadas do tipo 4 não são usadas.

   Apresente uma função (em pseudo-código) que recebe:

   - o número positivo N de tipos de tomadas/fichas que existem;
   - um vetor T (de comprimento N), com a quantidade de tomadas que existem de cada tipo;
   - um vetor F (de comprimento N), com a quantidade de fichas que existem de cada tipo;
   - uma matriz A (com um número positivo de linhas e duas colunas), com os tipos de adaptadores que existem.

   A função deve retornar o número mínimo de dispositivos que não podem ser ligados a uma tomada. **O corpo da sua função deve construir um grafo e chamar um ou vários algoritmos de grafos estudados, como se eles estivessem numa biblioteca**, mesmo que esses algoritmos retornem resultados que não interessam para resolver este problema e, consequentemente, sejam menos eficientes do que poderiam ser para este caso. Em vez de programar a construção do grafo, pode indicar claramente que grafo construiria, usando o exemplo para ilustrar a sua construção, e que estruturas de dados usaria para o implementar.