Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Verão 2016/17
Terceira série de problemas

Observações:

- Data de entrega: **19 de Junho de 2017**.
- **Para os métodos da primeira parte da série terão de ser desenvolvidos e entregues testes unitários.**

1 Exercícios

1. Realize a classe `TreeUtils`, contendo os seguintes métodos estáticos:

   1.1. O método

   ```
   public static Integer lower(Node<Integer> root, int k)
   ```

   que retorna o maior `Integer` i, presente na árvore binária de pesquisa com raíz `root`, que seja estritamente menor do que k, ou `null`, caso este não exista. Assuma que não existem elementos repetidos na árvore binária de pesquisa.

   1.2. O método

   ```
   public static int countLeavesAtLevel(Node<Integer> root, int k)
   ```

   que retorna o número de folhas na árvore binária de pesquisa com raíz `root` que se encontram no nível k.

   1.3. O método

   ```
   public static <E extends Comparable<E>> Node<E> createBST(E[] a)
   ```

   que retorna a referência para o nó raíz de uma árvore binária de pesquisa contendo os inteiros presentes no *array* `a`. A árvore resultante deve estar balanceada.

   Para as implementações destes métodos, considere que o tipo `Node<E>` tem 3 campos: um `value` e duas referências, `left` e `right`, para os descendentes respectivos.

2. Considere o seguinte excerto de código

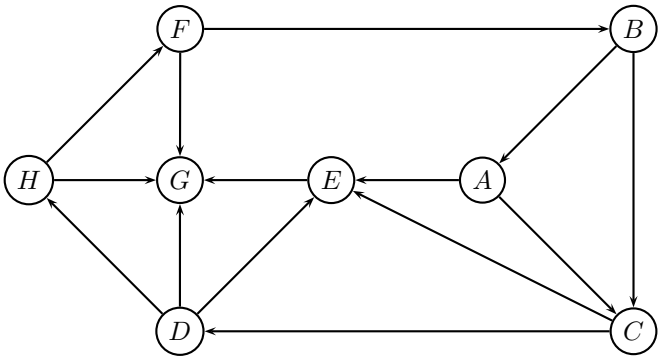
```
public class Xpto {
    private boolean[] marked;
    public Xpto(Graph G, int s) {
        marked = new boolean[G.nVertices()];
        xpto(G, s);
    }
    private void xpto(Graph G, int v) {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) xpto(G, w);
    }
    public boolean isMarked(int v) {
        return marked[v];
    }
}
```

em que G é um grafo orientado, `G.adj(v)` retorna uma sequência de vértices adjacentes ao vértice v e `G.nVertices()` retorna o número de vértices de G. Assuma que os vértices em G se encontram identificados de 0 a `G.nVertices()-1`.

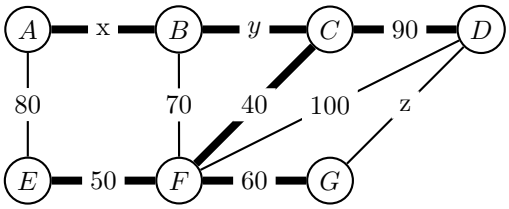
   2.1. Indique qual é o significado do retorno do método `isMarked`, para cada vértice do grafo, após criar um objeto do tipo `Xpto`.

2.2. Suponhamos que o objeto do tipo `Graph` é representado utilizando uma matriz de adjacências. Qual é o tempo de execução do construtor da classe `Xpto`?

3. Considere o seguinte grafo orientado com 8 vértices e 14 arcos. Qual a sequência de vértices visitados numa travessia em profundidade primeiro (DFS) sobre este grafo, com origem no vértice A ? Justifique. Considere que os vértices são visitados por ordem alfabética e que os vértices adjacentes de um vértice também são visitados por ordem alfabética.



4. Considere o seguinte grafo pesado, em que o peso dos arcos são inteiros. Assuma que os arcos sublinhados são os arcos que foram selecionados para uma árvore de menor abrangência (MST) segundo o algoritmo de Kruskal e o algoritmo de Prim (começando no vértice A). Deste modo, indique um valor possível para os pesos dos arcos x, y e z para o caso de um destes algoritmos. Identifique o algoritmo que selecionou e justifique a sua resposta.



2 Problema: A importância dos indivíduos nas redes sociais

Durante a última década tem existido um aumento do interesse no estudo e na extração de informação de redes complexas, tais como as redes sociais *Twitter* e *Facebook*. Estas redes são normalmente apresentadas sob a forma de grafos, nos quais os indivíduos são representados por vértices e os arcos correspondem a relações existentes entre dois indivíduos presentes na rede. Por exemplo, no caso do Twitter, a relação de seguidor (*follower*) é uma das relações que pode ser considerada para definir um arco.

De modo a compreender melhor estas redes, nomeadamente para efeitos de publicidade, são calculadas sobre as mesmas algumas métricas que permitem identificar vértices ou facilitar na localização de subgrupos.

Uma das métricas importantes é a medida de centralidade *Betweeneess*, aplicada aos vértices, a qual permite medir a importância do indivíduo na rede. Por definição, a medida de centralidade *"betweenness"* corresponde ao cálculo, para cada indivíduo, de quantas vezes esse mesmo indivíduo faz parte de um caminho mais curto entre outros dois utilizadores.

Outra medida de centralidade normalmente utilizada é a do *Degree (Grau)* de um vértice, isto é, o número de arcos que estão directamente conectados ao vértice

O problema é descrito por:

- um conjunto I de n indivíduos;
- uma lista L de ligações entre indivíduos, em que cada ligação é descrita por um par composto pela identificação de dois indivíduos.

O objectivo deste trabalho é portanto a realização de um programa que determine:

- o cálculo do *Degree* para todos indivíduos pertencentes à rede social;
- o cálculo da medida *Betweenneess*[1] para todos indivíduos pertencentes à rede social;

Funcionalidades a implementar

As funcionalidades a implementar são as seguintes:

1. Carregamento da informação da rede social, presente num ficheiro com formato *.edges*, dado o nome do ficheiro. As linhas descrevem as ligações de amizade existentes na rede, sendo estas constituídas por dois inteiros:
   1.1. identificador `i` de um amigo;
   1.2. identificador `j` de um amigo.
2. Comando que permite obter o cálculo do *Degree* para todos indivíduos pertencentes à rede social;
3. Comando que permite obter o cálculo da medida *Betweenneess* para todos indivíduos pertencentes à rede social;

Parâmetros de execução

Para iniciar a execução da aplicação a desenvolver, terá de o seguinte comando:

```
java centrality fileName.edges
```

o que compreende a funcionalidade 1. **Durante a sua execução**, a aplicação deverá processar os seguintes comandos:

- `g`
  que corresponde à funcionalidade 2.
- `b`
  que corresponde à funcionalidade 3.
- `e`
  que termina a aplicação.

Exemplo

Considere que o ficheiro de entrada, que designaremos por `exemplo.edges`, tem o seguinte conteúdo:

```
1 2
2 3
3 5
5 4
4 3
5 2
4 1
6 7
6 8
6 9
9 10
10 6
```

Um exemplo de execução da aplicação é a seguinte:

[1] Uma das variantes da definição de Betweenness

```
> java centrality exemplo.edges
> d
vertice degree
1 -> 2
2 -> 3
3 -> 3
4 -> 3
5 -> 3
6 -> 4
7 -> 1
8 -> 1
9 -> 2
10 -> 2

> b
vertice betweenneess
1 -> 1
2 -> 2
3 -> 1
4 -> 2
5 -> 0
6 -> 5
7 -> 0
8 -> 0
9 -> 0
10 -> 0
```

Relatório

O trabalho realizado deverá ser acompanhado de um relatório, que deverá incluir a avaliação experimental dos algoritmos desenvolvidos. Podem também recorrer à ferramenta **gephi** `https://gephi.org/` de modo a poderem ilustrar os exemplos que colocarem no relatório. Podem utilizar a classe `NumberFormat` para formatarem os resultados. Em anexo a esta série, encontra-se disponível um ficheiro para gerar testes.

ISEL, 1 de Junho de 2017.