Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Inverno 2021/22
Terceira série de problemas

Observações:

- Data de entrega: **27 de Janeiro de 2021**.
- **Podem ser utilizadas as estruturas de `java.util` ou `kotlin.collections` no Problema.**
- **Para os primeiros 3 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. A função

   ```
   fun <E> contains(root: Node<E>?, min:E, max:E, cmp:(e1:E, e2:E)->Int):Boolean
   ```

   que retorna `true` se e só se a árvore binária de pesquisa com raíz `root` contém algum elemento no intervalo `[min, max]`, segundo o critério de comparação `cmp`.

   1.2. A função

   ```
   fun createBSTFromRange(start:Int,end:Int): Node<Int>?
   ```

   que retorna a referência para o nó raíz de uma árvore binária de pesquisa contendo os inteiros presentes no intervalo fechado `[start,end]`. A árvore resultante deve estar balanceada.

   1.3. A função

   ```
   fun <E> isComplete(root: Node<E>?): Boolean
   ```

   que verifica se a árvore binária, referenciada por `root`, é completa, isto é?, todas as folhas da árvore binária se encontram no mesmo nível.

   Para as implementações destas funções, considere que o tipo `Node<E>` tem 3 propriedades: um `value`e duas referências `left` e `right`, para os descendentes respetivos.

2. Pretende-se realizar uma implementação do tipo de dados abstratos `graph`,que representa um grafo não orientado organizado através de listas de adjacência. Este tipo de dados é parametrizado pelos tipos genéricos `I` (tipo do identificador do vértice) e `D` (tipo dos dados associados ao vértice). A interface do tipo de dados `Graph` é da seguinte forma em Kotlin:

```
interface Graph<I, D>{
   interface Vertex<I, D> {
        val id: I
        val data: D
        fun setData(newData: D): D
        fun getAdjacencies(): MutableSet<Edge<I>?>
    }
    interface Edge<I> {
        val id: I
        val adjacent: I
    }
    val size: Int
    fun addVertex(id: I, d: D): D?
    fun addEdge(id: I, idAdj: I): I?
    fun getVertex(id: I): Vertex<I, D>?
    fun getEdge(id: I, idAdj: I): Edge<I>?
    operator fun iterator(): Iterator<Vertex<I, D>>
 }
```

Os componentes da interface `Graph` têm a seguinte funcionalidade:

- interface `Vertex`: representa um vértice de um grafo composto por um identificador e outros dados associados. Através da função `setData`, é possível atribuir novos dados ao vértice;
- função `getAdjacencies`: devolve um conjunto com todas as arestas adjacentes de um vértice (a ordem é indiferente). Se não existir nenhuma aresta devolve um conjunto vazio;
- interface `Edge`: representa uma aresta adjacente a um vértice. Uma aresta é composta pelo identificador do vértice origem e do vértice destino (adjacente);
- propriedade `size`: armazena o número de vértices existentes no grafo;
- função `addVertex`: adiciona um novo vértice ao grafo indicando o seu identificador e dados. Se o vértice já existir, devolve `null`sem adicionar, caso contrário, devolve os dados do vértice;
- função `addEdge`: adiciona uma nova aresta ao grafo indicando os identificadores dos vértices origem e destino (adjacente). Se o vértice origem ainda não existe devolve `null` sem adicionar, caso contrário, devolve o identificador do vértice adjacente;
- função `getVertex`: obtém um vértice dado o seu identificador. Se este não existir devolve null;
- função `getEdge`: obtém uma aresta dados os identificadores do vértice origem e destino (adjacente). Se esta não existir devolve null;
- função `iterator`: retorna um objeto `Iterator<Graph.Vertex<I,V>>` que permite a iteração dos vértices existentes no grafo.

2 Problema: Contar triângulos numa rede

A contagem de triângulos é essencial para avaliar o efeito de clustering em redes. Seja G um grafo que represente uma rede, por exemplo social, com n participantes e m pares de amigos. Então, é expectável que o número de triângulos seja muito maior neste grafo do que num grafo aleatório. O motivo é que se A e B são amigos e A é também amigo de C, então existe uma maior probabilidade, em geral, de que B e C sejam amigos. Deste modo, a contagem do número de triângulos é uma métrica que pode ser utilizada avaliar este efeito. A contagem de triângulos também pode ser utilizada para classificar a evolução temporal de uma rede, visto que à medida que a rede evolui, o número de triângulos se densifica.

O problema é descrito por:

- um conjunto V de n vértices;
- uma lista E de m ligações entre vértices, em que cada ligação é descrita por um par composto pela identificação de dois vértices.

O objetivo deste trabalho é portanto a realização de um programa que calcule o número de triângulos numa rede social e obter os `k` utilizadores que façam parte do maior número de triângulos existentes. **Cada grupo de alunos deverá escolher uma rede não orientada presente em `https://snap.stanford.edu/data/` e interpretar o estudo realizado. Note que cada rede tem um formato próprio. A rede a estudar deve ser diferente entre os vários grupos da turma e como tal deverá ser registada previamente em aula.**

Para ajudar na intepretaçao dos resultados, poderão consultar o livro: `https://www.cs.cornell.edu/home/kleinber/networks-book/`

Parâmetros de execução

Para executar a aplicação deverá realizar o comando `kotlinc k countingTriangles fileName`, em que `fileName` é o nome do ficheiro que contém a informação de uma rede e `k` o número de utilizares que façam parte do maior número de triângulos existentes

Exemplo

Considere que o ficheiro de entrada, que designaremos por `exemplo.txt` em que cada número é um identificador de um vértice e cada linha descreve uma aresta , 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, para `k=2` é a seguinte:

```
>kotlin 2 countingTrianglesKt exemplo.edges
> 3
> Vertices: 3;  5
```

Relatório

O trabalho realizado deverá ser acompanhado de um relatório, que deverá incluir a avaliação experimental e análise do espaço ocupado em memória.

ISEL, 14 de Dezembro de 2020.