isep Instituto Superior de Engenharia do Porto

# Estruturas de Informação

**Exame Época Normal**

23 de Janeiro 2020

**Departamento de Engenharia Informática**

Duração 2 horas

---

**Resolva cada exercício em folhas separadas**

**3.5 ptos**

**1.** Elabore um método que coloca numa lista de objetos Pair<Key,Value> a **fusão ordenada** de duas outras listas de objetos Pair<Key,Value>, **também ordenadas**. Os elementos de valor igual nas duas listas deverão ser colocados alternadamente na lista resultado. Por exemplo:

Lista A: [<A,2>, <A,2>, <A,5>] Lista B: [<B,1>, <B,1>, <B,2>, <B,3>, <B,4>, <B,4>, <B,5>]

**Lista Resultado**: [<B,1>, <B,1>, <A,2>, <B,2>, <A,2>, <B,3>, <B,4>, <B,4>, <A,5>, <B,5>]

Considere a seguinte assinatura

```
public static<K, E extends Comparable<E>> List<Pair<K,E>>
                              mergeLists(List<Pair<K,E>> A, List<Pair<K,E>> B)
```

**3.5 ptos**

**2.** Seja o seguinte código:

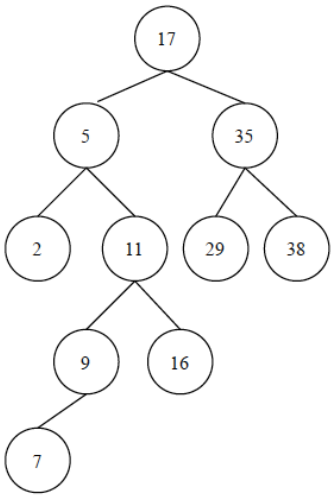
```
public static List<Character> misterio(Set<String> set) {
    List<Character> l = new LinkedList<>();
    boolean flag = true;
    int i = 0;

    while (flag) {
        flag = false;
        for (String s : set) {
           if (i < s.length()) {
             l.add(s.charAt(i));
             flag = true;  }
        }
        i++;  }
    return l;   }
```

**a)** Explique sucintamente o que o método faz.

**b)** Analise o método quanto à complexidade temporal, utilizando a notação Big-Oh. Justifique adequadamente.

**5 ptos**

**3.** Adicione à classe TREE<E> o método **inOrderSuccessor** que para um elemento fornecido devolve o elemento seguinte da árvore considerando a **travessia in-order**. Por exemplo para a árvore apresentada o elemento seguinte do 5 é o 7 (menor elemento da subárvore direita do nó 5), o elemento seguinte do 16 é o 17 e o elemento seguinte do 29 é o 35 (se o nó não tem subárvore direita, o seu sucessor é o primeiro antecessor que tem o nó à sua esquerda).



**Resolva cada exercício em folhas separadas**

**5 ptos**

**4.** Uma **rede de Petri** é um grafo orientado, composto por dois tipos de nós: transição e posição. No exemplo da Figura 1 os nós de transição são t1, t2, t3 e t4, enquanto que os nós de posição são p1, p2 e p3. Um nó de posição somente se pode ligar a nós de transição e um nó de transição somente se pode ligar a nós de posição.

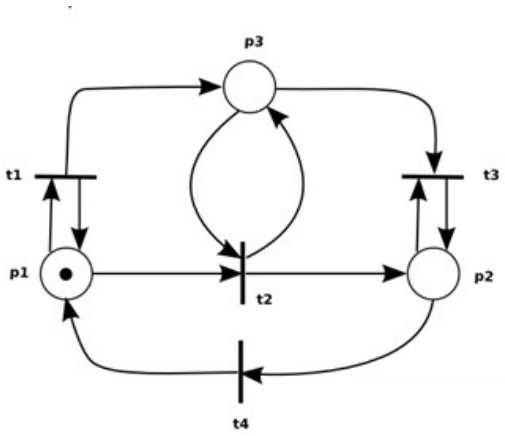



Figura 1

```
public interface  NoPetri {

   // verdadeiro se é nó transição, falso se é de
   posição
   public boolean isTransicao( );

   // verdadeiro se o nó de posição tem pelo menos
   um token
   public boolean temToken( );

   // remove um token de um nó de posição
   public void removeToken( );

   // adiciona um token a um nó de posição
   public void adicionaToken( );
}
```

Figura 2

Os nós de posição contêm zero ou mais tokens. No exemplo da Figura 1, somente o nó p1 tem um token, enquanto os outros se encontram vazios. Numa rede de Petri os **nós de transição** são **disparáveis** se todos os nós de posição que lhe ligam têm pelo menos um token. Na rede da Figura 1 somente o nó transição t1 é disparável. Considere a classe RedePetri que representa uma implementação de uma rede de Petri, usando a classe map de adjacências. Os nós são representados pela implementação da interface NoPetri (Figura 2). A classe RedePetri tem a seguinte definição.

```
public class RedePetri {
    private Graph<NoPetri,Integer> g = new Graph<>(true);
...
}
```

Crie uma função que devolva uma lista com todos os **nós de transição** de uma rede de Petri **disparáveis**. Considere que os métodos da interface NoPetri estão implementados.

**3 ptos**

**5.** Implemente um método que realize da forma **o mais eficiente possível** a fusão de duas heaps. O método deve retornar a heap resultante. Considere a seguinte assinatura:

```
public HeapPriorityQueue<Integer,String>
mergeHeaps(HeapPriorityQueue<Integer,String> hp1, HeapPriorityQueue<Integer,String> hp2)
```