Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Inverno 2017/2018
Segunda série de problemas
Parte I

Observações:

- Data de entrega: **13 de novembro de 2017**.
- Não é permitida a utilização de algoritmos e estruturas de dados já existentes na biblioteca base da plataforma Java.

1. Realize na classe **`Utils`** o método estático,

   `public static boolean verifyXML(String str)`

   que recebe uma string **`str`**, correspondente ao conteúdo de um documento XML, verificando se as etiquetas presentes em **`str`** se encontram emparelhadas e aninhadas corretamente. O método retorna **`true`** em caso afirmativo e **`false`** casocontrário. Considere para este exercício, a gramática especificada através da seguinte sintaxe, em EBNF[1]:

   ```
   Name ::= {letter | digit}+
   Space ::= {white character}+
   Element ::= '<' Name '>' . Description . '</' Name'>' |
   .           '<' Name '>' . {Space}* . Element . {Space}* . '</' Name'>'
   Description ::= {character | digit}*
   ```

   Exemplo de um documento XML:

   ```
   <catalog>
       <book id="bk101">
           <author>Gambardella, Matthew</author>
           <title>XML Developer's Guide</title>
           <genre>Computer</genre>
           <price>44.95</price>
           <publish date>2000-10-01</publish date>
           <description>An in-depth look at creating applications with XML.</description>
       </book>
   </catalog>
   ```

   Indique justificando, a complexidade do algoritmo.

2. Realize na classe **`ListUtils,`**

   2.1. O método estático,

   ```
   public static
   <E> Node<E> occurAtLeastKTimes(Node<E>[] lists, Comparator<E> cmp, int k)
   ```

   que dado um **`array`** de listas duplamente ligadas, não circulares e sem sentinela, ordenadas de modo crescente pelo comparador **`cmp,`** retorna uma lista duplamente ligada circular e com sentinela, ordenada de modo crescente pelo mesmo comparador, constituída pelos elementos que ocorram pelo menos **`k`** vezes nas listas presentes em **`lists.`** O *array* de listas **`lists`** pode ser modicado, assim como as respetivas listas.

---

[1] Neste enunciado é utilizada uma simplificação da notação EBNF (Extended Backus–Naur Form). Deste modo, descreve-se, de seguida, a notação utilizada:

Utilização	Notação
definição	::=
alternativa	|
repetição de 0 ou mais	{ … }*
repetição de 1 ou mais	{ … }+
concatenação	.
string terminal	‘ … ‘

Por exemplo, no caso do *array* de listas **lists** ser definido por [{1,2,2,5,6,6}, {0,0,2,2,6}, {6,6,7,7,7,7,8}], k=4 e o comparador **cmp** comparar os inteiros segundo a sua ordem natural, o método deverá retornar a lista constituída pelos seguintes elementos {2,6,7}.

2.2. O método estático,

```
public static <E> void internalReverse(Node<Node<E>> list)
```

que dada as listas duplamente ligadas, não circulares e sem sentinela presentes em **list,** inverte a ordem dos seus elementos respetivos. Note que a lista **list** é também uma lista duplamente ligada não circular e sem sentinela. Por exemplo, no caso da lista **list** ser definida por [{1, 2, 3}, {4, 7, 6}, {3, 1, 2, 4}], o método deverá transformar a lista **list** em [{3, 2, 1}, {6, 7, 4}, {4, 2, 1, 3}].

Para as implementações destes métodos, assuma que cada objeto do tipo **Node<E>** tem 3 campos: um **value** e duas referências, **previous** e **next.**

3. Realize na classe **Iterables**,

3.1. O método estático,

```
public static
Iterable<Integer> getSortedSubsequence(Iterable<Integer> src, int k)
```

que sendo **src** uma sequência não ordenada de inteiros, retorna um iterável com os elementos que pertençam à maior subsequência ordenada de **src** iniciada na primeira ocorrência de **k.** Por exemplo, no caso da sequência **src** ser definida por [1,3,2,5,7,6,4,8] e **k=5,** o método deverá retornar um iterável com os seguintes elementos [5,7,8].

3.2. O método estático,

```
public static
Iterable<Pair<String,Integer>> groupingEquals(Iterable<String> words)
```

que dada a sequência de palavras **words**, ordenada de modo crescente por ordem lexicográfica, retorna uma sequência de pares, em que cada par é composto por uma palavra distinta presente em **words** e pelo número de ocorrências dessa mesma palavra. Para a implementação deste método, considere que a classe Pair está definida do seguinte modo:
public class **Pair<E,F>{ public E first; public F second; }.**

As implementações destes métodos devem minimizar o espaço ocupado pelo iterável. O iterador associado ao iterável retornado não suporta o método **remove.**