Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Verão 2021/22
Segunda série de exercícios

Observações:

- Data de entrega: **14 de Maio de 2022**.
- É apenas permitida a utilização de estruturas de dados presentes em `java.util` ou `kotlin.collections` na implementação dos exercícios 2 e no ponto 1 do problema.
- O convite ao *assignment* associado a esta série de problemas no *github classroom* será enviado no contexto da respetiva turma.

1 Exercícios

1. Realize a função

   `fun median(v:IntArray, l:Int, r:Int):Int`

   que retorna a mediana dos elementos do *sub-array* (v, l, r). A *mediana* é o valor que separa a metade maior e a metade menor de um conjunto de dados. No entanto, se a dimensão do conjunto for par, a mediana é definida como a média dos dois valores do meio.

2. Assuma que se pretende definir um tipo de dados `IntArrayList` que lhe permite armazenar uma lista de `k` inteiros, em que `k` é conhecido previamente. Além do construtor, que inicializa este tipo de dados, deverá suportar as seguintes operações, garantindo que a complexidade das mesmas é `O(1)`:

   - `append(x:Int):Boolean` que adiciona o inteiro `x` à lista;
   - `get(i:Int):Int?` que permite obter o `i`-ésimo elemento desta lista; Retorna `null` caso não exista.
   - `addToAll(x:Int)` que adiciona a todos os inteiros presentes nesta lista o inteiro `x`.

   Implemente o tipo de dados `IntArrayList` e justifique a respetiva implementação.

3. Realize as seguintes funções:

   3.1. `fun <E> getMiddle(list:Node<E>):Node<E>`
   que, dada a lista duplamente ligada, sem sentinela e não circular, referenciada por list, retorna o nó do meio da lista. Caso a lista seja de dimensão par, retorna o primeiro nó do meio.

   3.2. `fun <E> removeAfterIntersectionPoint(list1:Node<E>, list2:Node<E>, cmp:Comparator<E>)`
   que, dadas duas listas duplamente ligadas, circulares e com sentinela, referenciadas por `list1` e `list2`, remove de `list1` os nós a partir do nó de interseção de ambas, caso exista. O nó de interseção de duas listas é o primeiro nó após o qual ambas as listas partilham a mesma sequência de valores segundo o comparador `cmp`. Por exemplo, no caso das listas serem: `list1 = {3,5,2,7,4}` e `list2 = {9,3,10,8,2,7,4}`, existe o nó de interseção entre as duas listas que é o que contém o valor 2. Neste caso, após a execução do método a `list1` ficará `list1 = {3,5}`.

   3.3. `fun <E> bubbleSort(list: Node<E>, cmp: Comparator<E>)`
   que dada a lista duplamente ligada, circular e com sentinela, referenciada por `list`, realiza a ordenação de modo crescente, segundo o comparador `cmp`, dos elementos presentes em `list`, segundo o algoritmo bubblesort.

4. Pretende-se realizar uma implementação do tipo de dados abstratos `HashMap`, que representa uma coleção de pares chave-valor, suportado numa tabela de dispersão (hash table) com encadeamento externo em listas duplamente ligadas, não circulares e sem sentinela. Este tipo de dados é parametrizado pelos atributos `K` (tipo das chaves) e `V` (tipo dos valores). Assuma que a função de dispersão a utilizar, é a função `hashCode` definida para os objetos do tipo `K`. Para a implementação deste tipo de dados, deve definir o tipo `HashNode<K,V>`, contendo quatro campos: um key do tipo `K`; um value do tipo `V`; e as referências `previous` e `next` do tipo `HashNode<K,V>`. A interface do tipo de dados `HashMap` é representada da seguinte forma em `Kotlin`:

```
interface MutableMap<K,V> {
  interface MutableEntry<K, V>{
         val key: K
         val value:V
         fun setValue(newValue: V): V
 }
val size: Int
fun put(key: K, value: V): V?
fun remove(key: K): V?
operator fun get(key: K): V?
fun iterator(): Iterator<MutableEntry<K, V>>
}
```

Deste modo, a interface `MutableMap` contém:

- A interface `MutableEntry` representa um par chave-valor associados, sendo também possível atribuir um novo valor a uma chave já existente através da método `setValue`;
- Propriedade `size`: armazena o número de elementos presentes no mapa;
- Função `put`: associa `value` a `key` no mapa. Retorna o valor anteriormente associado a `key`, ou `null` caso a chave não exista anteriormente no mapa;
- Função `remove`: remove do mapa a chave especificada e o seu valor correspondente;
- Operador `get`: devolve o valor associado à chave ou `null` caso este não exista. Este operador permite que a sintaxe `map[key]` possa ser utilizada;
- Função `iterator`: retorna um objeto `Iterator<MutableEntry<K, V>>` que permite a iteração dos pares chave-valor existentes no mapa. Os elementos podem ser iterados por qualquer ordem. O tipo de dados `hashMap` também deverá suportar:
- A função de extensão `map(transform:(MutableMap.MutableEntry <K,V>)->R): List<R>` que retorna uma lista de valores resultantes da aplicação da função de transformação transform a cada entrada no mapa original. Nesta implementação poderá utilizar o tipo `ArrayList` de `kotlin.collections`.
- A função de extensão `filter(predicate:(MutableMap.MutableEntry<K,V>)->Boolean): List<MutableMap.MutableEntry<K,V>>` que retorna uma nova lista contendo todos os pares chave-valor validados pelo predicado `predicate`. Nesta implementação poderá utilizar o tipo `ArrayList` de `kotlin.collections`.

2 Problema: Filtrar palavras distintas de uma determinada dimensão

Pretende-se desenvolver uma aplicação que permita filtrar as palavras distintas, de uma determinada dimensão, presentes em um documento de texto. Para iniciar a execução da aplicação `FilterWordsKt`, terá de ser executado:

```
kotlin FilterWordsKt document.txt
```

Durante a sua execução, a aplicação deverá processar os seguintes comandos:

- `allWords`, que lista todas as palavras distintas que ocorram no ficheiro de entrada.
- `withDim k`, que lista todas as palavras distintas que ocorram no ficheiro de entrada com dimensão `k`.
- `exit`, que termina a aplicação.

Implementação

1. Primeira Implementação: realize esta aplicação, utilizando as estruturas presentes em `kotlin.collections` ou `java.util` que considere necessárias.
2. Segunda Implementação: realize esta aplicação, sem utilizar as estruturas presentes em `kotlin.collections` ou `java.util`.

Avaliação Experimental Realize uma avaliação experimental do(s) algoritmo(s) desenvolvido(s) para a resolução deste problema. Apresente os resultados graficamente, utilizando uma escala adequada. Compare os mesmos com a complexidade assintótica esperada da solução.

Relatório O trabalho realizado no contexto desta secção deverá ser acompanhado de um relatório, que deverá incluir a avaliação experimental.

ISEL, 12 de Abril de 2022.