Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Verão 2024/2025
Segunda série de problemas

Observações:

- Data de entrega: **11 de maio de 2025**.
- Não é permitida a utilização de estruturas de dados presentes em `java.util` ou `kotlin.collections` exceto na primeira implementação do problema**.**
- Valorizam-se soluções mais eficientes.

I. Exercícios

Para teste dos algoritmos desta secção, considere o conjunto de testes unitários fornecidos com o projeto.

1. Realize a seguinte função :

```
fun minimum(maxHeap: Array<Int>, heapSize: Int): Int
```

que retorna o menor elemento do *max heap* representado pelo parâmetro `maxHeap`. A dimensão do *heap* é indicada por `heapSize`. O algoritmo deve tirar partido das propriedades de um *max heap* na procura do menor elemento.

2. Definição do tipo de dados `IntArrayList`

Pretende-se definir um tipo de dados `IntArrayList` que armazena uma lista de `k` inteiros, onde `k` é conhecido previamente. A estrutura de dados deve garantir que todas as operações tenham complexidade *O(1)*.
A lista deve seguir uma disciplina FIFO (*First-In*, *First-Out*) no tratamento dos elementos, ou seja, os elementos são removidos pela ordem de inserção.
O tipo de dados `IntArrayList` deve suportar as seguintes operações:

- `append(x: Int): Boolean` - Adiciona o inteiro x ao final da lista. Retorna `true` se a operação for bem-sucedida e `false` caso a lista esteja cheia.
- `get(n: Int): Int?` - Retorna o enésimo elemento da lista ou `null` caso o índice seja inválido.
- `addToAll(x: Int)` - Adiciona `x` a todos os inteiros presentes nesta lista.
- `remove(): Boolean` - Remove o primeiro elemento da lista. Retorna `true` se a operação for bem-sucedida e `false` caso a lista esteja vazia.

3. Realize as seguintes funções :

3.1. `fun splitEvensAndOdds(list: Node<Int>)`

Dada a lista duplamente ligada com sentinela e circular referenciada por `list` reorganiza a lista de modo que todos os números pares fiquem consecutivos no início da lista.
Por exemplo, caso `list` seja {7, 1, 2, 5, 4, 8, 3, 21}, após a execução da função, os números pares devem ficar consecutivos no início, como em {8, 4, 2, 5, 1, 7, 3, 21}, sendo este um exemplo da lista resultante que a função pode produzir.

3.2. `fun <T> intersection(list1: Node<T>, list2: Node<T>, cmp: Comparator<T>): Node<T>?`

Dadas duas listas duplamente ligadas, circulares e com sentinela, referenciadas por `list1` e `list2`, e ordenadas de modo crescente segundo o comparador `cmp`, a função retorna uma nova lista com os elementos que estejam simultaneamente presentes em ambas as listas, removendo-os de `list1` e `list2`.
A lista retornada deverá ser duplamente ligada, não circular e sem nó sentinela, ordenada de modo crescente, e sem elementos repetidos. Deve ainda reutilizar os nós de uma das listas (`list1` ou `list2`).

Para as implementações destas funções, assuma que o tipo `Node<T>` tem 3 campos: um `value` do tipo `T` e duas referências, `previous` e `next`, para o elemento anterior e seguinte.

4. Implementação do tipo de dados abstratos `MutableMap<K,V>`

Implemente uma estrutura de dados do tipo abstrato `MutableMap<K, V>`, que representa um contentor associativo de pares chave-valor. A estrutura de dados deve ser baseada numa tabela de dispersão (*hash table*) com encadeamento externo, usando listas ligadas, não circulares e sem nó sentinela. Este tipo de dados é parametrizado pelos atributos `K` (tipo das chaves) e `V` (tipo dos valores).
Para a implementação, considere que a função de dispersão utilizada será a função `hashCode` definida para os objetos do tipo `K`.
Cada par chave-valor armazenado na tabela de dispersão deve ser representado pelo tipo `HashNode<K,V>`, contendo três campos: um `key` do tipo `K`; um `value` do tipo `V`; e uma referência `next` do tipo `HashNode<K,V>`.

A interface do tipo de dados *hash map* é representada da seguinte forma em Kotlin:

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

Os componentes da interface `MutableMap` têm as seguintes funcionalidades:

- `MutableEntry`: Representa um par chave-valor, permitindo atribuir um novo valor a uma chave já existente através do método `setValue`;
- Propriedade `size`: Representa o número de elementos presentes no mapa;
- Propriedade `capacity`: Indica a dimensão da tabela de dispersão;
- Função `put`: Associa `value` à `key` no mapa. Retorna o valor anteriormente associado a `key`, ou `null` caso a chave não exista no mapa;
- Operador `get`: Devolve o valor associado à chave ou `null` caso este não exista. Este operador permite que a sintaxe `map[key]` seja utilizada;
- Função `iterator` (presente na interface `Iterable<...>`): Retorna um objeto `Iterator<MutableEntry<K, V>>`, permitindo a iteração sobre os pares chave-valor existentes no mapa. Os elementos podem ser iterados por qualquer ordem.

A tabela deve ser expandida para o dobro quando o número de elementos multiplicado pelo *fator de carga* for igual ou superior à capacidade da tabela. O fator de carga é a razão entre o número de elementos presentes na tabela de dispersão e a sua dimensão.

II. Problema: Operações entre coleções de pontos no plano

Pretende-se desenvolver uma aplicação para realizar operações entre coleções de pontos no plano, nomeadamente as operações de união, interseção e diferença. Cada coleção de pontos encontra-se descrita num ficheiro de texto. Cada ponto no ficheiro é descrito por um identificador e duas coordenadas: X e Y. A aplicação `ProcessPointsCollections` deve apresentar as seguintes funcionalidades:

- Recebe como parâmetro dois ficheiros de texto (com extensão `.co`).
- Permite produzir um novo ficheiro (com extensão `.co`) contendo os pontos, sem repetições, que ocorram em pelo menos um dos ficheiros de entrada (operação `union`).
- Permite produzir um novo ficheiro (com extensão `.co`) contendo os pontos comuns entre ambos os ficheiros de entrada (operação `intersection`).
- Permite produzir um novo ficheiro (com extensão `.co`) contendo os pontos únicos que estejam presentes apenas em um dos ficheiros de input (operação `difference`).

Parâmetros de execução

Para iniciar a execução da aplicação `ProcessPointsCollections`, terá de se executar:

kotlin `ProcessPointsCollections`

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

- `load document1.co document2.co` - Carrega os pontos dos dois ficheiros num único *hash map* ou tabela de dispersão, que deverá ser estruturado de forma a permitir a consulta eficiente para responder a todas as operações subsequentes.

- `union output.co` - Produz o ficheiro `output.co` contendo os pontos presentes em pelo menos um dos ficheiros de entrada, sem repetições.
- `intersection output.co` - Produz o ficheiro `output.co` contendo os pontos que ocorrem em ambos os ficheiros de entrada, sem repetições.
- `difference output.co` - Produz o ficheiro `output.co` contendo os pontos que ocorrem no ficheiro `document1.co` mas que não ocorram no ficheiro `document2.co`, sem repetições.
- `exit` - Termina a aplicação.

Formato dos ficheiros com extensão “.co”

Cada linha dos ficheiros com extensão .co pode ser de um dos seguintes 3 tipos:

- coordenadas de um ponto - compostas pelo caracter 'v', seguido do identificador do ponto, da sua coordenada em ordem a X e da sua coordenada em ordem a Y.
- comentário - iniciado pelo caractere 'c'.
- problema - iniciado pelo caractere 'p'.

No contexto deste trabalho, as linhas do tipo 'c' e 'p' não são relevantes.

Implementação

1. Primeira implementação: Use as estruturas presentes na Kotlin Standard Library que considere necessárias.
2. Segunda implementação: Utilize a estrutura de dados implementada na questão I.4.

Avaliação Experimental

Realize uma avaliação experimental do(s) algoritmo(s) desenvolvido(s) para a resolução deste problema. Como exemplo, poderá utilizar os ficheiros que se encontram disponíveis juntamente com este enunciado. Apresente os resultados graficamente, utilizando uma escala adequada.