Algoritmos e Estruturas de Dados
2014/15
Primeiro Teste – 29 de outubro de 2014
Departamento de Informática, Universidade Nova de Lisboa

Atenção: Os Anexos ao teste poderão ser-lhe úteis.

1. Considere o método `alg` apresentado abaixo.

```
//Requires: n!= null && pos >=0 && pos <= n.length
protected static int alg(int n[], int pos){

   if (pos == n.length)
      return 0;
   else return n[pos] + alg(n, ++pos);
}
```

   Determine a complexidade temporal do método `alg`, no melhor caso, no pior caso e no caso esperado, **justificando** todos os cálculos com muita clareza.

2. Desenvolva o método recursivo `countDigit` que, dado um número inteiro (superior ou igual a 0) e um dígito (entre 0 e 9), devolve o número de ocorrências do dígito no número. Uma vez desenvolvido o método recursivo pedido, determine a complexidade temporal do método `countDigit`, no melhor caso, no pior caso e no caso esperado.

```
//Requires: n >= 0 && digit >= 0 && digit <= 9
public static int countDigit(int n, int digit){
   …
}
```

   Exemplo: `a execução de countDigit(389004005, 0) devolve como resultado 4, enquanto que a execução de countDigit(38004005, 7) devolve 0.`
   Nota: Implemente a solução em código Java. Apenas as soluções recursivas de `countDigit` serão avaliadas.

3. O Tipo Abstrato de Dados `TransportData` define as operações associadas à gestão dos cartões de transporte de uma empresa de transporte de autocarros. Um cartão de transporte (TAD `TransportCard`) tem um saldo em Euros associado (para simplificar assumimos que todos os valores numéricos são inteiros). Este saldo pode diminuir com viagens do utilizador do cartão e aumentar graças a carregamentos efetuados pelo mesmo. O cartão contém ainda os últimos 10 movimentos efetuados sobre o cartão (viagens e carregamentos). O sistema a desenvolver, suportado pelo TAD `TransportData` permite inserir novos cartões no sistema, removê-los, pagar viagens e recarregar cartões, além de possibilitar a iteração dos últimos 10 movimentos de um determinado cartão, pela ordem em que estes foram efetuados, iniciando-se no movimento mais antigo e terminando no mais recente. Apresentam-se abaixo os interfaces para os Tipos Abstratos de Dados `TransportData` e `TransportCard`.

Relativamente ao problema proposto:

- Explicite detalhadamente as estruturas de dados e as variáveis de instância mais adequadas para implementar a interface TransportData (e todas as que dela dependerem);
- Descreva brevemente como implementaria as cinco operações e calcule as suas complexidades temporais, no caso esperado, justificando. **Não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações de acordo com a sua escolha de estruturas de dados.**

```
interface TransportCard{
   // Devolve o identificador único do cartão.
   String getCardId();

   // Devolve o saldo atual do cartão
   int getCardBalance();

   // Insere um gasto efetuado no cartão, como resultado de uma viagem.
   // O gasto é executado em Euros (inteiros) e só é possível se o cartão
   // tiver saldo suficiente. Se o saldo não for suficiente devolve false.
   // Requires: tripValue >= 1
   boolean spend(int tripValue);

   // Efetua o carregamento do saldo do cartão com o valor value.
   // Requires: value >= 1
   void reload(int value);

   // Devolve um iterador com os últimos 10 movimentos do cartão. A iteração
   // deve ser executada por ordem cronológica de execução dos movimentos.
   Iterator<Integer> lastMovements();
}

public interface TransportData{

   // Cria cartão com identificador cardId e saldo 0 (zero).
   void insertCard(String cardId) throws ExistingCardException;

   // Remove o cartão e devolve o saldo atual do cartão cardId.
   int removeCard(String cardID) throws NonExistingCardException;

   // Insere um gasto tripValue efetuado no cartão cardId, se o saldo
   // do cartão for suficiente.
   void spendCard(String cardId, int tripValue)
      throws InvalidValueException, NonExistingCardException,
         NoBalanceException;

   // Efetua o carregamento do saldo do cartão cardId com o valor value.
   void reloadCard(String cardId, int value)
      throws InvalidValueException, NonExistingCardException;

   // Devolve um iterador com os últimos 10 movimentos do cartão cardID,
   // por ordem cronológica.
   Iterator<Integer> lastMovementsCard(String cardID)
      throws NonExistingCardException;
}
```