Algoritmos e Estruturas de Dados
Exame de Recurso
Departamento de Informática, Universidade Nova de Lisboa
15 de Janeiro de 2015

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

1. a) Considere a árvore AVL apresentada na Figura 1. Em cada nó está apenas representada a chave do mesmo.

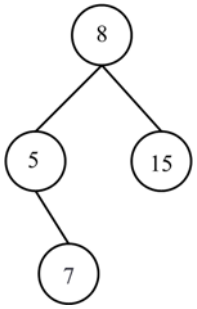

Figura 1

Desenhe a árvore AVL resultante de remover o elemento com a chave 8 na árvore apresentada;

a) Considere agora a árvore AVL apresentada na Figura 2. Desenhe a AVL resultante de inserir um elemento com chave 30.

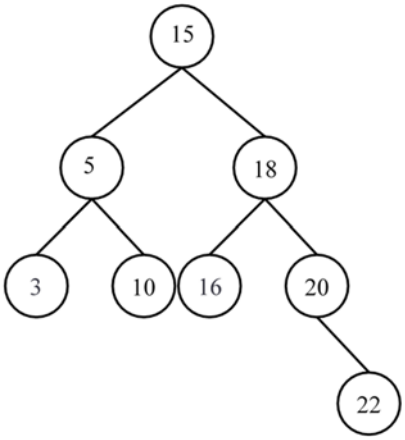

Figura 2

(O exame continua na página 2)

2. Implemente a classe `VectorIterator<E>` que permite iterar, um a um, todos os elementos (`<E>`) contidos num vetor de Listas (interface `List<E>`) . O construtor desta classe irá receber um vetor de listas e deverá preparar esta estrutura para uma iteração dos elementos `E` contidos nas listas guardadas em cada uma das posições do vetor. O iterador a desenvolver deverá começar por iterar todos os elementos da lista associada à posição `0` do vetor (se esta não estiver vazia), depois os elementos da lista na posição `1` e por aí a fora, até chegar à última posição do vetor. Algumas listas poderão estar vazias. O esqueleto da classe a implementar está abaixo. Lembre-se que o construtor de um iterador deve sempre chamar o método `rewind()`.
   Calcule ainda complexidade temporal de cada um dos métodos a implementar, no pior caso, justificando. Para este cálculo pode assumir que o número de posições do vetor de listas será menor que o número total de elementos do tipo `E` guardados no vetor (no conjunto total das listas). Pode ainda assumir que as listas utilizadas em cada posição do vetor são instâncias da classe `DoublyLinkedList<E>`.

```
public class VectorIterator<E> implements Iterator<E> {

      List<E>[] v;
      public VectorIterator(List<E>[] l) {
            …
            this.rewind();
      }

      public void rewind(){
            …
      }
      public E next() throws NoSuchElementException {
            …
      }
      public boolean hasNext() {
            …
      }

}
```

(O exame continua na página 3)

3. Considere a Classe `DoublyLinkedList<E>` (ver anexo) apresentada nas aulas teóricas. Implemente um método para esta classe chamado `Merge()` que executa a fusão da lista da instância da classe, com uma lista recebida como parâmetro. A fusão deverá ser executada usando um elemento de cada lista por ordem, começando pela lista da instância. A sua implementação deve ter em conta casos especiais, como o que fazer quando uma ou as duas listas envolvidas estão vazias e deve funcionar para listas com diferentes números de elementos. Calcule ainda a complexidade temporal do método `Merge()` no pior caso, justificando.

   Exemplo: Considere duas listas duplamente ligadas *L1* e *L2*, contendo elementos do tipo inteiro. Se *L1* contiver os elementos 4, 3, 7 e 2 por esta ordem e *L2* contiver os elementos 1 e 5, também pela ordem dada, poderemos executar a seguinte chamada:

   L1`.merge(`*L2*`).`

   Depois de executado o método, a lista *L1* conterá os seguintes elementos, pela ordem dada: 4, 1, 3, 5, 7, 2. A lista *L2* ficará vazia.

```
public class DoublyLinkedList<E>
   implements List<E> {

   // Node at the head of the list.
   protected DListNode<E> head;

   // Node at the tail of the list.
   protected DListNode<E> tail;

   // Number of elements in the list.
   protected int currentSize;

   ...

   // Merge list l with this.
   public void merge(DoublyLinkedList<E> l){
   ...
   }

}
```

4. O twitter vai eleger a melhor série televisiva de sempre. Para tal, os utilizadores registados vão poder votar nas suas séries favoritas. Cada utilizador só tem direito a uma única votação, mas nessa votação pode incluir várias séries. A única limitação é que uma votação não pode exceder os 140 caracteres. Por exemplo, o utilizador @iTweetALot votou nas séries @CommunityTV e @GameOfThrones. Considere que tanto os utilizadores como as séries são identificadas pela respectiva conta twitter. Proponha uma implementação para o Tipo Abstrato de Dados (TAD) BestTVShowEver. Neste TAD é possível: registar uma votação; devolver a série mais votada até ao momento; eliminar o voto de um utilizador numa série específica; devolver o número total de votos recebidos; listar as votações de um utilizador. O TAD em questão é apresentado abaixo:

```
public interface BestTVShowEver {

   //Regista a votação do utilizador user em todas as séries
   //contidas na lista favorites. Pode assumir que a soma dos
   //caracteres de todas as Strings contidas na lista é <= 140.
   void addVote(String user, List<String> favorites)
      throws NotRegistered, AlreadyVoted;

   //Devolve a série de televisão mais votada até ao momento
   String bestShow() throws NoVotes;

   //Elimina a votação do utilizador user na série show
   void clearVote(String user, String show)
      throws NotRegistered, NonExistingVote;

   // Devolve o número total de votos registados
   int getTotalNumberOfVotes()

   //Devolve um iterador que percorre, por qualquer ordem, os
   //votos do utilizador user
   Iterator<String> listVotes(String user)
      throws NotRegistered, NonExistingVote;

}
```

Baseando-se na descrição apresentada e no interface fornecido, explicite detalhadamente as estruturas de dados e as variáveis de instância mais adequadas para implementar o mesmo, descreva brevemente como implementaria as cinco operações e calcule as suas complexidades temporais, no caso esperado, justificando. Se a sua tarefa for facilitada pela criação de TADs auxiliares à sua solução, deverá também descrever as variáveis de instância e estruturas de dados associadas a estes na sua resposta. **Não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações de acordo com a sua escolha de estruturas de dados.**

5. Considere o desenvolvimento de um sistema de gestão de mensagem de texto dentro de uma grande organização. O sistema contempla a existência de utilizadores que recebem mensagens de colegas da empresa. Dado que estas são mensagens de trabalho, as mensagens não são privadas mas todas têm apenas um emissor e um destinatário. Assim, todos os utilizadores podem enviar mensagens, mas podem também recebê-las. Desta forma, cada *utilizador* irá ter acesso a informação sobre todas as mensagens que enviou, assim como todas as que recebeu, se estas não tiverem sido arquivadas. Além disso, um utilizador irá ser identificado pelo seu identificador. Dado que este é um sistema de gestão, é importante que exista uma estrutura que guarde todas as mensagens alguma vez geradas, tendo estas sido ou não arquivadas por determinado utilizador. O acesso global às mensagens pode ser feito via o identificador da mensagem. Cada *mensagem* conterá assim, além do seu identificador, informação sobre o seu emissor, o destinatário, a data/hora em que foi gerada e o texto. Cada utilizador pode,

em qualquer momento, listar as mensagens que enviou ou recebeu, por ordem da mais recente para a mais antiga. Pode ainda decidir arquivar todas mensagens (enviadas ou recebidas). Esta decisão de arquivo é apenas realizada para o utilizador que a efetuou e não tem reflexos nem nas mensagens guardadas pelos outros utilizadores, nem no gestor global de mensagens. As operações suportadas pelo sistema a desenvolver são assim as seguintes:

a) *Inserir um Utilizador*, dando o seu identificador e nome. A operação só tem sucesso se o identificador do utilizador ainda não existir no sistema;
b) *Enviar mensagem,* dando o identificador da mensagem, o seu texto, o identificador do emissor e o identificador do destinatário. Esta operação só terá sucesso se o identificador da mensagem não existir no sistema, e se tanto o emissor como o destinatário existirem no sistema. A data da mensagem será a hora de sistema;
c) *Consultar utilizador*, dando o seu identificador. Esta operação só tem sucesso se o utilizador existir no sistema e deverá devolver o nome do mesmo;
d) Consultar mensagem, dando o seu identificador. Esta operação só tem sucesso se a mensagem tiver sido alguma vez enviada e irá devolver o texto da mensagem, o nome do emissor, o nome do destinatário e a data de envio;
e) *Arquivar lista de mensagens*, dando o identificador do utilizador e a indicação de qual a lista de mensagens do mesmo a arquivar (enviadas ou recebidas). Esta operação arquiva todas as mensagens da lista do utilizador e só tem sucesso se o utilizador já existir no sistema. A partir do momento de arquivamento, a lista arquivada fica vazia. Este arquivamento não tem consequências para as listas de mensagens dos outros utilizadores;
f) *Listar mensagens enviadas por utilizador,* dando o seu identificador. Esta operação só terá sucesso se o identificador do utilizador emissor existir no sistema. As mensagens deverão ser apresentadas por ordem da mais recente para a mais antiga. Os dados a apresentar, para cada mensagem, são o identificador do destinatário, o texto e a data de envio;
g) *Listar mensagens recebidas por utilizador,* dando o seu identificador. Esta operação só terá sucesso se o identificador do utilizador destinatário existir no sistema. As mensagens deverão ser apresentadas por ordem da mais recente para a mais antiga. Os dados a apresentar, para cada mensagem, são o identificador do emissor, o texto e a data de envio;

Espera-se que o número de utilizadores seja da ordem das centenas e que o número de mensagens seja da ordem dos milhares. Com base nesta especificação, explicite detalhadamente as estruturas de dados e as variáveis de instância mais adequadas para implementar esta aplicação, descreva sumariamente os algoritmos para efetuar as 7 operações (enumeradas de (a) a (g)) e calcule (justificando) as suas complexidades temporais, no caso esperado. Se a sua tarefa for facilitada pela criação de TADs auxiliares à sua solução, deverá também descrever as variáveis de instância e estruturas de dados associadas a estes na sua resposta. **Não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações de acordo com a sua escolha de estruturas de dados.**