Exame de Algoritmos e Estruturas de Dados

Departamento de Informática

Universidade Nova de Lisboa

17 de Janeiro de 2008

1. [3.5 valores] Implemente o método *numberOfNodesWith1Child*, na classe *BinarySearchTree*, e programe todos os métodos auxiliares desta classe de que necessitar. Calcule a complexidade temporal do seu algoritmo, no melhor caso, no pior caso e no caso esperado, justificando.

```
public class BinarySearchTree<K extends Comparable<K>, V>
    implements OrderedDictionary<K,V>
{
    // The root of the tree.
    protected BSTNode<K,V> root;

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

    . . . . . .

    // Returns the number of nodes in the tree that have exactly one child.
    protected int numberOfNodesWith1Child( );
}
```

2. [3.5 valores] Considere o tipo abstracto de dados *Fila com Dois Níveis* de elementos do tipo E, caracterizado pela interface *TwoLevelQueue*.

```
public interface TwoLevelQueue<E>
{
    // Inserts the specified element at the rear of the queue,
    // classifying it as a level one element.
    void enqueueLevel1( E element );

    // Inserts the specified element at the rear of the queue,
    // classifying it as a level two element.
    void enqueueLevel2( E element );

    // Removes and returns the element at the front of the queue.
    E dequeue( ) throws EmptyQueueException;

    // Returns an iterator of the level one elements in the queue
    // (in proper sequence).
    Iterator<E> iteratorLevel1( );

    // Returns an iterator of the level two elements in the queue
    // (in proper sequence).
    Iterator<E> iteratorLevel2( );
}
```

Explicite **detalhadamente as estruturas de dados** mais adequadas para implementar este tipo abstracto de dados, descreva brevemente como implementaria as cinco operações e calcule as suas complexidades temporais, no melhor caso, no pior caso e no caso esperado, justificando.

3. [3.5 valores] Considere a função *max/2*, que calcula o máximo de um vector de elementos do tipo E, de acordo com o comparador fornecido, pela técnica da divisão e conquista.

```
static <E> E max( E[] vector, Comparator<E> comp )
    throws EmptyArrayException
{
    if ( vector.length == 0 )
        throw new EmptyArrayException();

    return max(vector, comp, 0, vector.length − 1);
}


// Precondition: firstPos <= lastPos.
static <E> E max( E[] vector, Comparator<E> comp, int firstPos, int lastPos )
{
    if ( firstPos == lastPos )
        // The vector has just one element.
        return vector[firstPos];
    else
    {
        // The vector has at least two elements.
        int middlePos = ( firstPos + lastPos ) / 2;
        E maxLeft = max(vector, comp, firstPos, middlePos);
        E maxRight = max(vector, comp, middlePos + 1, lastPos);
        return maxOf2(maxLeft, maxRight, comp);
    }
}


static <E> E maxOf2( E element1, E element2, Comparator<E> comp )
{
    if ( comp.compare(element1, element2) >= 0 )
        return element1;
    else
        return element2;
}
```

Determine a complexidade temporal do método *max/2* quando este é chamado com um vector de n elementos (com $n \geq 1$), no melhor caso, no pior caso e no caso esperado, assumindo que a complexidade da comparação de dois elementos do tipo E é sempre constante. Justifique todos os cálculos com muita clareza.

(Continue, porque o exame tem mais duas perguntas.)

4. [5 valores] Pretende-se desenvolver uma aplicação que permita efectuar consultas sobre as mensagens que passam por um servidor de *e-mail*.

   Cada *mensagem* possui um *assunto*, um *remetente*, uma sequência não vazia de *destinatários*, uma *data* de envio e um *corpo*. Quando o servidor recebe uma mensagem, associa-lhe uma *estampilha*, única no sistema, que permite definir a ordem de chegada das mensagens ao servidor.

   A aplicação deve permitir efectuar as seguintes operações.

   (a) Adicionar uma mensagem no sistema, tendo a mensagem (com todos os seus dados) e a estampilha (que se sabe ser maior do que qualquer estampilha presente no sistema).

   (b) Listar todas as mensagens com um dado assunto.
   Informação por mensagem: remetente, sequência de destinatários e data.
   A listagem deve aparecer por ordem crescente de estampilha.

   (c) Listar todos os assuntos das mensagens enviadas por um dado remetente, indicando, para cada assunto, o número de mensagens com esse assunto enviadas pelo remetente.
   Informação por item: o assunto e o número de mensagens com esse assunto enviadas pelo remetente.
   A listagem deve aparecer por ordem alfabética de assunto.

   (d) Listar todas as mensagens com um dado assunto que foram enviadas para um dado destinatário.
   Informação por mensagem: remetente, data e corpo.
   A listagem deve aparecer por ordem crescente de estampilha.

   Espera-se que o número de assuntos não exceda 25 000, que o número de remetentes e o número de destinatários não ultrapassem 15 000 e que o número de mensagens não seja superior a 500 000.

   Explicite **detalhadamente as estruturas de dados** mais adequadas para implementar esta aplicação, descreva sumariamente os algoritmos para efectuar as quatro operações (enumeradas de (a) a (d)) e calcule (justificando) as suas complexidades temporais, no caso esperado.

(Continue, porque o exame tem mais uma pergunta.)

5. [4.5 valores] Considere o método *eliminateRepetitions*, na classe *DoublyLinkedList*, que elimina todos os elementos repetidos da lista ligada, deixando a primeira ocorrência (no sentido da cabeça para a cauda).

   Por exemplo, se l_1 e l_2 forem listas duplamente ligadas com os elementos:

   $$17,\ 44,\ 17,\ 10,\ 5,\ 10 \quad \text{e} \quad 11,\ 11,\ 11\,,$$

   após l_1.eliminateRepetitions() e l_2.eliminateRepetitions(), l_1 e l_2 ficam, respectivamente, com:

   $$17,\ 44,\ 10,\ 5 \qquad \text{e} \qquad 11\,.$$

```
class DListNode<E>
{
    private E element;                  // Element stored in the node.
    private DListNode<E> previous;      // (Pointer to) the previous node.
    private DListNode<E> next;          // (Pointer to) the next node.

    public DListNode( E theElement ) { ··· }
    public DListNode( E theElement, DListNode<E> thePrevious,
       DListNode<E> theNext ) { ··· }

    public E getElement( ) { ··· }
    public DListNode<E> getPrevious( ) { ··· }
    public DListNode<E> getNext( ) { ··· }
    public void setElement( E newElement ) { ··· }
    public void setPrevious( DListNode<E> newPrevious ) { ··· }
    public void setNext( DListNode<E> newNext ) { ··· }
}

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;

    ..........................................

    // Eliminates all repetions from the list,
    // keeping the first occurence of equal elements.
    public void eliminateRepetitions( );
}
```

Implemente o método *eliminateRepetitions* (na classe *DoublyLinkedList*) e programe todos os métodos auxiliares desta classe de que necessitar. Pode assumir que os elementos da lista são comparáveis entre si (E extends Comparable<E>). Calcule a complexidade temporal do seu algoritmo, no caso esperado, justificando.