Recurso de Algoritmos e Estruturas de Dados

Departamento de Informática

Universidade Nova de Lisboa

12 de Fevereiro de 2008

1. [3.5 valores] Decidiu-se implementar árvores binárias cujos nós possuem um apontador para o pai (para além do elemento e dos apontadores para os filhos esquerdo e direito). O problema é que os algoritmos de inserção e de remoção destas árvores não estão a afectar correctamente o apontador para o pai. Felizmente, a raiz e o número de elementos da árvore, bem como todos os apontadores para os filhos, estão correctos.

   Considere então as classes *BTNode*, dos novos nós das árvores binárias de elementos do tipo E, e *BinaryTree*, das novas árvores binárias de elementos do tipo E, ambas no pacote *dataStructures*.

```
class BTNode<E>
{
    private E element;                  // Element stored in the node.
    private BTNode<E> leftChild;        // (Pointer to) the left child.
    private BTNode<E> rightChild;       // (Pointer to) the right child.
    private BTNode<E> parent;           // (Pointer to) the parent.

    public BTNode( E theElement ) { this(theElement, null, null, null); }
    public BTNode( E theElement, BTNode<E> theLeft, BTNode<E> theRight,
        BTNode<E> theParent )
    {   element = theElement;  leftChild = theLeft;
        rightChild = theRight;  parent = theParent;
    }
    public E getElement( ) { return element; }
    public BTNode<E> getLeft( ) { return leftChild; }
    public BTNode<E> getRight( ) { return rightChild; }
    public BTNode<E> getParent( ) { return parent; }
    public void setElement( E newElement ) { element = newElement; }
    public void setLeft( BTNode<E> newLeft ) { leftChild = newLeft; }
    public void setRight( BTNode<E> newRight ) { rightChild = newRight; }
    public void setParent( BTNode<E> newParent ) { parent = newParent; }
}

public class BinaryTree<E>
{
    protected BTNode<E> root;           // The root of the tree.
    protected int currentSize;          // Number of elements in the tree.
    ...........................................
    protected void restoreAllParentPointers( );
}
```

   Implemente o método *restoreAllParentPointers* (na classe *BinaryTree*), que afecta correctamente o apontador para o pai de todo o nó da árvore, 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.

2. [4.5 valores] Sejam S uma sequência de elementos do tipo E e $f : \mathrm{E} \longrightarrow \mathrm{Bool}$ uma função booleana (chamada *filtro*). A *filtragem* de S por f, que se denota por $S \downarrow f$, é a maior sub-sequência de S cujos elementos e verificam $f(e) = \mathit{true}$.

   Por exemplo, se $p : Z \longrightarrow \mathrm{Bool}$ for a função que vale *true* quando o número é par:

   - $(1,\ 2,\ 3,\ 4,\ 4,\ 3,\ 2,\ 1) \downarrow p$ é $(2,\ 4,\ 4,\ 2)$;
   - $(5,\ 5,\ 3,\ 5,\ 1,\ 2) \downarrow p$ é (2);
   - $(71,\ 13) \downarrow p$ é a sequência vazia; e
   - a filtragem da sequência vazia por p é a sequência vazia.

   Suponha que, em termos de programação, os filtros (de elementos do tipo E) são caracterizados pela interface *Filter* e ambas as sequências são geradas por iteradores (de elementos do tipo E).

```
public interface Filter<E>
{
    // Returns true iff the specified element passes through the filter.
    boolean accepts( E element );
}
```

   Programe a classe *FilterIterator* cujo construtor recebe um iterador da sequência original e um filtro, e devolve um iterador da filtragem da sequência pelo filtro. Defina as variáveis de instância e programe os quatro métodos seguintes.

```
public class FilterIterator<E> implements Iterator<E>
{
    // Returns an iterator of S ↓ filter,
    // where S is the sequence generated by the specified iterator.
    public FilterIterator( Iterator<E> sequence, Filter<E> filter );

    // Returns true iff the iteration has more elements.
    // In other words, returns true if next would return an element
    // rather than throwing an exception.
    public boolean hasNext( );

    // Returns the next element in the iteration.
    public E next( ) throws NoSuchElementException;

    // Restarts the iteration.
    // After rewind, if the iteration is not empty,
    // next will return the first element in the iteration.
    public void rewind( );
}
```

   Calcule (justificando) as complexidades temporais dos seus quatro métodos, no melhor caso e no pior caso, assumindo que a complexidade da função *accepts* é sempre constante.

3. [3.5 valores] Considere o tipo abstracto de dados caracterizado pela interface *TimeSet*.

```
public interface TimeSet<E extends Comparable<E>>
{
    // If the set does not contain the specified element, inserts it in the set
    // as permanent or temporary (according to the isPermanent parameter)
    // and returns true; otherwise, returns false.
    boolean insert( E element, boolean isPermanent );

    // Makes the (existent) specified element permanent.
    void makePermanent( E element ) throws NoSuchElementException;

    // Makes the (existent) specified element temporary.
    void makeTemporary( E element ) throws NoSuchElementException;

    // Returns: 0, if the specified element does not belong to the set;
    // 1, if it is a permanent element; and 2, if it is a temporary element.
    int contains( E element );

    // If the set contains the specified element, removes it from the set and
    // returns true; otherwise, returns false.
    boolean remove( E element );

    // Returns an iterator of all elements in the set.
    Iterator<E> iterator( );

    // Returns an iterator of the temporary elements in the set,
    // which preserves the element order relation.
    Iterator<E> iteratorTemp( );
}
```

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

4. [3.5 valores] A função *catalanRec*, que calcula o n-ésimo número de Catalan pela recorrência clássica (com $n \geq 0$), tem complexidade exponencial. Apresente um algoritmo polinomial que calcule o mesmo valor, aplicando a técnica da função-memória ao algoritmo dado. Determine a complexidade temporal e espacial do seu algoritmo, no melhor caso e no pior caso, justificando todos os cálculos com muita clareza.

```
public static long catalanRec( int n )
{
    if ( n == 0 )
        return 1;
    else
    {
        long sum = 0;
        for ( int i = 0;  i < n;  i++ )
            sum = sum + catalanRec(i) * catalanRec(n − i − 1);
        return sum;
    }
}
```

5. [5 valores] Pretende-se desenvolver uma aplicação para gerir as tarefas (em execução e pendentes) numa dada organização. Cada *tarefa* tem um *número* único que a identifica, um *responsável*, uma *data* prevista de conclusão e um conjunto de *nomes de aplicações* que vai utilizar.

   Existe um limite máximo para o número de tarefas *em execução*, que é de 5 000. Sempre que se mandar executar uma tarefa e esse limite tiver sido alcançado, a tarefa fica *pendente*. As tarefas pendentes entram imediatamente em execução, por ordem de chegada, à medida que as tarefas em execução forem terminando.

   A aplicação deve permitir efectuar as seguintes operações.

   (a) Mandar executar uma tarefa, dando o número, o responsável, a data e os nomes das aplicações.

   (b) Dar uma tarefa (em execução) por terminada, dando o seu número.

   (c) Calcular o número de tarefas em execução.

   (d) Listar, por ordem alfabética, os nomes das aplicações utilizadas por uma tarefa em execução, dado o seu número.

   (e) Listar, por ordem alfabética, os nomes das aplicações utilizadas pelas tarefas em execução, indicando, para cada aplicação, as tarefas que a utilizam, por ordem de entrada em execução.
   Informação por tarefa: número, responsável e data.

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