Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software

Examen Final, Septiembre 2015

1.[3 ptos] Los **números de Jacobsthal** son una sucesión de números enteros definida de la siguiente manera:

$$\begin{array}{lcl} J(0) & = & 0 \\ J(1) & = & 1 \\ J(n) & = & J(n-1) + 2 * J(n-2) \text{ si } n > 1 \end{array}$$

Dada la siguiente especificación de un algoritmo que calcula el n-ésimo número de Jacobsthal:

$\{n \geq 0\}$
fun Jacobsthal(**int** n) **return int** x
$\{x = J(n)\}$

donde la función $J : \mathbb{N} \to \mathbb{N}$ es la definida anteriormente, diseñar y verificar (o derivar) un algoritmo iterativo que dado un natural $n \geq 0$ calcule $J(n)$ en tiempo en $O(n)$. Justificar adecuadamente el coste del algoritmo.

2. [3 ptos] En una matriz de N filas y M columnas hay un rectángulo desde una posición desconocida (*fila*, *columna*) ($0 \leq$ *fila* $< N$, $0 \leq$ *columna* $< M$) hasta la posición $(N-1, M-1)$ que contiene el número 1 en todas sus posiciones. El resto de posiciones de la matriz contienen el número 0. Se pide implementar un algoritmo eficiente que encuentre la posición (*fila*, *columna*). Justificar adecuadamente el coste del algoritmo.

Por ejemplo, si $N = 5$, $M = 4$, y la matriz de entrada es:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

el algoritmo debe devolver $(3, 1)$.

3. [2,5 ptos] Implementa la siguiente función:

```
template <typename T>
bool coinciden(const Arbin<T> &a, const List<T> &pre);
```

que diga si la lista `pre` coincide con el recorrido en preorden del árbol `a`. No puedes hacer uso de estructuras de datos auxiliares (arrays, pilas, colas, listas, árboles etc.) ni pedir/liberar memoria adicional (hacer `new` o `delete`).

4. [1,5 ptos] Implementa la siguiente función:

```
template <typename T>
void invierteBase(Stack<T> &p, int n);
```

que reciba la pila `p` y un valor entero `n` ($0 \leq n < p.size()$) y modifique `p`, de forma que los n valores de la cima queden en el orden que están y los $p.size() - n$ valores del fondo de la pila queden en orden inverso al de entrada.

En la implementación de la función deben utilizarse únicamente TADs vistos en clase, se valorará que se seleccionen los TADs más apropiados para la resolución del problema. No se pueden utilizar los arrays de C++. Se valorará la eficiencia de la solución propuesta.