Escuela de Ingeniería Informática
Universidad de Oviedo

Algoritmia
2015-2016

Control 2 – 18 de abril de 2016

Apellidos, nombre ________________________________ NIF: ______________

Pregunta 1 (4 puntos)

Queremos crear un algoritmo para resolver el juego del Sudoku. Para diseñarlo utilizaremos un algoritmo que utiliza la técnica de **Backtracking**.

a) (2,5 p.) Rellena los huecos para que el algoritmo de backtracking proporcione una solución:

```
public class SudokuPrimera {
	static int n;                     // Tamaño del Sudoku
	static int[][] tablero; // Tablero para el juego del Sudoku
	static boolean solEncontrada;

	static void backtracking(________________________________) {
		// x==n no quedan posiciones libres --> solución
		if (x==n) {
			________________________________
			mostrarTablero();
		}
		else
			for (________________________________) {
	if(____________________________________________________________
		________________________________) {
					________________________________

					int[] posVacia = new int[2];

					________________________________
				backtracking(________________________________);

					________________________________
			}
		}
	}

	/* Comprueba si se puede poner el número k en la fila x */
	static boolean comprobarFila(int x, int k) { … }

	/**
	 * Comprueba si se puede poner k en la columna y
	 */
	static boolean comprobarColumna(int y, int k) {
		boolean b = true;
		________________________________
		________________________________
		return b;
	}
```

```
static boolean comprobarRegion(int x, int y, int k) {
      boolean b = true;
      ____________________________
      ____________________________

      for (____________________________)
            for (____________________________)
                  if (tablero[i][j] == k) b=false;
      return b;
}

static void mostrarTablero() {…}
 /* Busca la siguiente posición vacía para poner un número */
static int[] buscarVacia(int x, int y) {…}

}
```

b) (0,5 p.) ¿Cuántos hijos genera cada estado en este problema? Explica tu respuesta.

c) (1 p.) Trabajando con backtracking, normalmente tenemos dos tipos diferentes de complejidades. ¿Cuáles son estas complejidades? Explica claramente cuando obtenemos cada una de ellas haciendo referencia al tipo de árbol generado.

Pregunta 2 (3 puntos)

La función de *Ackermann* es un ejemplo clásico de función recursiva, es notable especialmente porque no es una función con recursión primitiva. En computabilidad fue un contraejemplo a lo que se creía en 1900 que cada función computable era recursiva primitiva. Esta función se utiliza como benchmark para medir la habilidad de los compiladores para optimizar la recursión.

Normalmente se define como:

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m-1, A(m,n-1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

a) (0,5 p.) Escribe el código fuente en Java de un método que resuelva el problema utilizando un esquema recursivo para cualquier valor de *m* y *n*.

b) (2 p.) Diseña un algoritmo para resolver Ackermann para *A(3,5)* usando la estrategia de programación dinámica. No hace falta escribir el código, crea la tabla que necesitamos y rellena los valores. ¿Cuál es el resultado final?

c) (0,5 p.) Explica las principales diferencias entre Divide y Vencerás y Programación dinámica tanto en el diseño como en la complejidad.

Apellidos, nombre _____ NIF: _____

Pregunta 3 (3 puntos)

En una compañía que vende productos de madera se ha creado un sistema que permite construir listones de cualquier longitud (múltiplo del listón pequeño) a partir de pequeños listones prefabricados de determinadas medidas que disponen de un sistema en encaje entre sí.

El proveedor sólo suministra un conjunto de listones con unas longitudes dadas a elegir entre los dos tipos siguientes, del que se puede encargar el número que sea necesario:

1. Conjunto 1: Se dispone de listones prefabricados con las siguientes longitudes: 10, 30 y 40,90.
2. Conjunto 2: Se dispone de listones prefabricados con las siguientes longitudes: 10, 20, 40, 80.

Teniendo en cuenta esto se pide:

a) (1 p.) Indica un heurístico que podamos utilizar en un algoritmo voraz para calcular qué listones necesitamos y describe cómo se aplica.
b) (2 p.) Qué conjunto de listones encargará la compañía si quiere utilizar siempre la menor cantidad de listones, independientemente de la longitud del listón a construir. Demuéstralo con ejemplos.