Examen de Teoría de la Programación

E. U. ING. TEC. EN INFORMÁTICA DE OVIEDO

Final Septiembre – Curso 2005-2006 11 de septiembre de 2006

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

3. (1,5 puntos) El algoritmo de Floyd consiste en encontrar el camino más corto entre todos los pares de nodos de un grafo. Dado un grafo G, lo que queremos es hallar el camino más corto para ir desde un nodo *i* hasta otro nodo *j* de forma directa o a través de otro nodo *k*.
Para resolver este problema partimos de la matriz de pesos del grafo (D), es decir, una matriz que recoge el coste de ir de un nodo a otro del grafo sin pasar por nodos intermedios. Cuando no hay camino directo entre dos nodos, la celda correspondiente tendrá valor infinito. Y se aplica la siguiente función D(i,j)= min(D(i,j), D(i,k)+D(k,j)) pudiendo ser k cualquier otro nodo del grafo.

a) **Razonar por qué decimos que Floyd se encuadra dentro de las técnicas de programación dinámica.**
b) **Representar gráficamente el array necesario para implementar el algoritmo, suponiendo que tenemos un grafo de 4 nodos.**
c) **Explicar que valores se pueden rellenar de forma directa en la tabla.**
d) **Marcar sobre el array del apartado b) las casillas de las que depende (3,2) para calcular su valor.**
e) **¿Qué complejidad tiene este algoritmo?**

(Problema expuesto en el grupo B)

4. (2 puntos) El problema del salto del caballo consiste en recorrer un tablero de ajedrez completo (8x8) a base de movimientos como los que realiza el caballo. Se quieren buscar todas las formas posibles de recorrer el tablero por la técnica de backtracking (vuelta atrás).
Las casillas a las que puede acceder un caballo desde una dada se muestran en la tabla adjunta.

	5		4	
6				3
		*		
7				2
	8		1	

Dada la clase *SaltoCaballo* que se muestra a continuación:

a) **Escribir el método que realiza el backtracking, que llamaremos *ensayar*.**
b) **Escribir el método main() que hace la llamada a *ensayar*.**

```
public class SaltoCaballo
{
      // Desplazamiento vertical del caballo
      private int[] despVertical=   {2, 1, -1, -2, -2, -1, 1, 2};
      // Desplazamiento horizontal del caballo
      private int[] despHorizontal= {1, 2, 2, 1, -1, -2, -2, -1};

      private int tam;                        // Tamaño del tablero
      private int[][] tablero;  // Estructura de datos para representar el estado

      /** Constructor */
      public SaltoCaballo(int iniTam)
      {     ...    }

      /** Método que permite mostrar una solución */
      public void mostrarTablero()
      {     …      }
}
```

(Problema expuesto en el grupo C)

5. (1,5 puntos) Un turista desea realizar una ruta por 6 capitales europeas. El recorrido está establecido pero el turista podrá elegir cual es la ciudad origen, con lo cual queda determinado donde finalizará su recorrido. Cada ciudad tiene un coste, que serán los euros que cuesta pasar allí una noche. El turista puede recorrer a lo sumo 2 de estas ciudades en un día, con lo cual deberá parar a dormir o bien en la primera que visite o visitar 2 y quedarse a dormir en la última.
El turista desea obtener el recorrido que cumpla las condiciones establecidas y que tenga coste mínimo. Disponemos de la clase Ciudad con los métodos públicos necesarios para acceder y modificar estos datos:

```
public class Ciudad {
  private String nombre;
  private int peso;
  private boolean visitado;
…
}
```

a) **Razona por qué se debería utilizar un algoritmo devorador.**
b) **Describir un heurístico que proporcione una solución óptima.**
c) **Pseudocódigo del algoritmo devorador.**

(Problema expuesto en el grupo A)

6. (2 puntos) El problema del puzzle se desarrolla sobre un tablero de 16 posiciones, donde hay colocadas 15 fichas, quedando una posición vacía. Las fichas no se pueden levantar del tablero, por lo que sólo es posible su movimiento por medio de desplazamientos sobre el mismo.

Tabla 1. Estado final

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

El objetivo del juego es, a partir de un estado inicial dado, obtener un estado objetivo (el mostrado arriba, tabla 1). Los únicos movimientos válidos son aquellos en que una ficha adyacente a la posición libre, se mueve hacia ésta. Se pueden ver estos movimientos permitidos como un desplazamiento de la casilla vacía hacia una de sus cuatro vecinas.
Queremos resolver este problema mediante la técnica de ramificación y poda. Y utilizamos el siguiente heurístico de ramificación: Calcular el número de piezas que están en una posición distinta de la que les corresponde en la disposición final.
Se pide:

a) **Representa gráficamente el árbol de estados de la técnica ramifica y acota que partiendo del estado que damos a continuación (tabla 2) desarrolla dos de sus nodos.**
b) **Representa gráficamente los estados que quedan en la cola de prioridad después de realizar las operaciones del apartado anterior.**
c) **Explica las ventajas para este problema en concreto, de haber empleado esta técnica en vez de backtracking.**
d) **Justifica si hay alguna posibilidad de desarrollar algún heurístico de poda. Si es así proporciona uno y comenta brevemente como se implementaría.**

Tabla 2. Estado inicial para el problema

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12