Examen de Teoría de la Programación

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

Final Junio – Curso 2005-2006 12 de junio de 2006

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

Grupo al que asistes en teoría: _____

3. (1 punto) La condición que tienen que cumplir los elementos de un montículo de mínimos es que el padre siempre ha de ser menor o igual que los hijos.

a) Definir mediante lógica de predicados la invariante de clase que permita asegurar esta condición para la clase Montículo. Sabiendo que el montículo tiene *n* nodos, almacenados en el array *a* y para el nodo en la posición *i* (comenzando por 1) los hijos están en las posiciones 2*i* y 2*i*+1.

a es el array del montículo y contiene n nodos.

$\{ \forall i \in \{1..n\}.\ 2i \in \{1..n\} \rightarrow a[i] \leq a[2i] \wedge 2i+1 \in \{1..n\} \rightarrow a[i] \leq a[2i+1] \}$

b) Queremos definir una operación que reordene los elementos del array del montículo, de tal forma que si hacemos un recorrido del primero al último estén ordenados de forma creciente, para ello, en la clase Java, tenemos un método publico ordenarCreciente(), que a su vez llama a un método privado ordenaSiguiente(int i), que ordena cada uno de los elementos del array. ¿De qué forma aplicarías la invariante a estos dos métodos (escribe el esqueleto de los métodos y la llamada a la invariante)?

```
public ordenarCreciente()
{
        …
        // Comprobación invariante al final del método público
        assert esMonticuloMinimos();
                /* Supongo que tengo un método: esMonticuloMinimos(),
                    donde implemento el predicado establecido en el apartado
a */
}

private ordenaSiguiente(int i)
{
        …
        // No hay que comprobar nada ya que es un método privado
}
```

4. (1,5 puntos) En el mundial de fútbol que se está celebrando en Alemania, los 32 equipos que participan están divididos en grupos de 4, que se enfrentan entre si en una liguilla. A diferencia de otros campeonatos en forma de liguilla, aquí no hay dos enfrentamientos sino uno entre cada pareja de equipos, ya que todos juegan "fuera" de casa.
Si aplicamos el planteamiento del, mal llamado, problema del *torneo de tenis*, los enfrentamientos quedarían así:

Equipo	Día 1	Día 2	Día 3
1	2	(n/2)+1= 3	(n/2)+1+1= 4
2	1	(n/2)+1+1=4	((n/2)+1)+(1+1)%(n/2)=3
3	(1,1)+n/2= 4	1	2
4	3	2	1

a) Especificar que características se plantean en la solución de este problema para que podamos considerarlo como un caso divide y vencerás.

En este problema es difícil dar la solución directa al problema completo; sin embargo es muy sencillo solucionar casos pequeños como el enfrentamiento de dos equipos. Tenemos un esquema recursivo de división del problema en otros más pequeños hasta llegar al caso base.

b) Cuál es el caso base y cómo se resuelve.

2 equipos y la solución es que se enfrentan entre sí.

c) Rellena la tabla (la que está abajo) para los emparejamientos si en vez de grupos de 4 equipos hubiese grupos de 8.

d) Marca las necesarias para tener los emparejamientos simples.

Equipo	Día 1	Día 2	Día 3	Día 4	Día 5	Día 6	Día 7
1	2	(n/2)+1= 3	(n/2)+1+1= 4	5	6	7	8
2	1	(n/2)+1+1=4	((n/2)+1)+(1+1)%(n/2)=3	6	7	8	5
3	(1,1)+n/2= 4	1	2	7	8	5	6
4	3	2	1	8	5	6	7
5	6	7	8	1	4	3	2
6	5	8	7	2	1	4	3
7	8	5	6	3	2	1	4
8	7	6	5	4	3	2	1

(Hay más variantes correctas para rellenar la tabla)

5. (1,25 puntos) Dado un sistema monetario compuesto por monedas de distinto valor, el problema del cambio consiste en descomponer en monedas de curso legal cualquier cantidad dada *L*, utilizando el menor número posible de monedas de dicho sistema monetario.

Siendo *n* el número de tipos de monedas distintos, *L* la cantidad que queremos descomponer y T(1..n) un vector con el valor de cada tipo de moneda del sistema. Suponemos que tenemos una cantidad inagotable de monedas de cada tipo. Se debe calcular la cantidad mínima final de monedas de cada tipo para dicha cantidad *L* inicial.

La función *C(i,j)* es el número mínimo de monedas para obtener la cantidad *j* restringiéndose a los tipo T(1), T(2), … T(i).

$$C(i,j)=\begin{cases}\infty \quad si \quad i=1\wedge 1\le j<T(i)\\ 0 \quad si \quad j=0\\ 1+C(i,j-T(i)) \quad si \quad i=1\wedge j\ge T(i)\\ C(i-1,j) \quad si \quad i>1\wedge j>T(i)\\ Min\{C(i-1,j),1+C(i,j-T(i))\} \quad en \quad otro \quad caso\end{cases}$$

Utilizaremos la técnica de Programación Dinámica para obtener la solución al problema. Suponemos que para ganar la eliminatoria un equipo debe de ganar tres partidos.

a) Representar gráficamente el array necesario para implementar el algoritmo, señalando el significado de filas y columnas (no hace falta rellenarlo). Tener en cuenta que queremos calcular la probabilidad antes de comenzar la eliminatoria.

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

b) Plantear el orden correcto para rellenar la tabla.

Se puede rellenar de muchas formas, por ejemplo por filas de arriba abajo.

c) Dado el sistema monetario euro construir y rellenar la tabla para devolver un cambio de 7 céntimos.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	1	2	3	4	5	6	7
2	1	1	2	2	3	3	4
5	1	1	2	2	1	2	2

Las celdas de la tabla contienen, sólo hay que aplicar la función dada, el mínimo número de monedas, tanto del tipo correspondiente a la fila como de las anteriores, para devolver la cantidad correspondiente a la columna actual.

(Problema expuesto en el grupo B)

6. (1,75 puntos) El algoritmo de Prim permite, dado un grafo conexo no dirigido, donde cada arista posee una longitud no negativa, calcular el árbol de recubrimiento mínimo (o árbol abarcador). Este árbol está formado por un subconjunto de las aristas del grafo dado, tal que utilizando este subconjunto todos los nodos queden conectados y además la suma de las longitudes de las aristas de este subconjunto es mínima.
Prim parte de un nodo cualquiera del grafo y tiene como función heurística buscar la arista más corta que parte de este o de uno de los nodos ya conectados para llegar a un nodo no conectado del árbol de recubrimiento mínimo actual. Esta operación se repite hasta que tengamos todos los nodos del grafo inicial conectados.

a) Explicar qué características del algoritmo de Prim permiten encuadrarlo dentro de los algoritmos voraces.

Se basa en una función heurística que guía al algoritmo en la selección del nodo adecuado. Esto evita recorrer un gran número de estados en el árbol.
Se elige el siguiente estado con información local: la arista más corta que conecta el grafo. Esta decisión no se revoca nunca, no hay vuelta atrás.

b) Escribir pseudocódigo para la función heurística.

Buscar la arista más corta que conecte nodos (conectados) que ya pertenecen al árbol de recubrimiento mínimo con nodos que todavía no pertenecen (no conectados).

c) Escribir el pseudocódigo del método y su llamada que permita obtener la solución a este problema mediante el algoritmo descrito. Para simplificar se puede suponer la existencia de funciones auxiliares para trabajar con los nodos del grafo describiendo brevemente (una línea) las operaciones de cada una.

(Problema expuesto en el grupo C)

```
public prim(Grafo grafo)
{
        int i= 0;
        int numNodos= grafo.numNodos();
        int arbolRecubrimiento[]= new int[numNodos];

        arbolRecubrimiento[i++]= 1;    // nodo 1 como inicial
        // mientras no hayamos conectado todos los nodos
        while (i<numNodos)
        {
                // Heurístico:
                // Buscar la arista más corta
                // que parte de uno de los nodos ya conectados
                // para llegar a un nodo no conectado

                // Marcar la arista como utilizada
                // Añadir el nuevo nodo como conectado
                arbolRecubrimiento[i]= nuevo nodo;
```

```
                i++;

        }
}
```

7. (1,5 puntos) Dado un tablero de ajedrez de tamaño n x n, un rey es colocado en una casilla arbitraria de coordenadas (x, y). El problema consiste en determinar los (n^2-1) movimientos de la figura de forma que todas las casillas del tablero sean visitadas una sola vez, si tal secuencia existe.

a) Cuántos hijos teóricos tiene cada nodo en el árbol que representa la exploración del problema y a que se corresponde cada uno de ellos.

8 hijos teóricos, tantos como movimientos posibles tiene la pieza del rey en ajedrez.

b) Qué condiciones permitirán determinar que un estado es posible.

Para que un estado sea posible el movimiento no debe dejar el rey fuera de los límites del tablero, ni que caiga en una celda por la que ya se pasó.

c) Qué se debe anotar en cada estado.

Se anota la nueva casilla que ocupa el rey y las coordenadas de esta para partir de ella en el siguiente movimiento.

d) Que condición indicará que hemos alcanzado la solución.

Que hemos alcanzado el movimiento n^2, lo que indicará que hemos cubierto todas las casillas del tablero.

e) Escribe la cabecera del método Java explicando los parámetros utilizados y escribe como se realiza la llamada inicial en el método main() con las inicializaciones que fuesen necesarias.

```
Public recorridoRey(int numMovimiento, int x, int y)

numMovimiento: es el número de movimiento que realizamos dentro del tablero
x: coordenada x dentro del tablero
y: coordenada y dentro del tablero

recorridoRey(1,1,1);
En el primer movimiento partimos de la posición (1,1) del tablero.
```

(Problema expuesto en el grupo A)