# Examen de Teoría de la Programación

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

Final Febrero – Curso 2008-2009 7 de febrero de 2009

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

**3. (2 puntos) Se pretende resolver mediante la técnica de backtracking el problema de los eslabones. En este problema, se pretende construir una cadena de longitud *n* cm, a partir de eslabones de 4 longitudes diferentes (disponibles en número ilimitado), que se encuentran almacenados en un vector de enteros de 4 posiciones. Se pretende que el número de eslabones utilizado sea el mínimo posible.**

**a) (1,5 puntos) Completar el código Java para dar solución a este problema (escribirlo en los recuadros preparados a tal efecto).**

```
public class Eslabones
{
// Longitudes eslabones
private static int eslabones[] = { 1, 12, 5, 25 };

// solución parcial
private int aux[] = new int[eslabones.length];

// Solución óptima
private long nEslabonesOpt = 100000;// num. eslabones de la solución
private int solOptima[] = new int[eslabones.length]; // de cada tipo

public void buscarEslabones(long n,int cEslab) {
      // Bucle para desarrollar cada hijo
      for (int i = 0; i < eslabones.length; i++) {
            if (eslabones[i] <= n) {  // comprueba si estado es válido
                  aux[i]++; // Incremente eslabones de ese tipo

                  if ((n-eslabones[i]) != 0) {  // si no es solución
                        buscarEslabones(n-eslabones[i],cEslab+1);
                  }
                  else {
                        // si es solución --> comprobamos si es
                        // mejor que la optima hasta el momento
                        if (cEslab < nEslabonesOpt) {
                              nEslabonesOpt = cEslab;
                              solOptima = (int[]) aux.clone();
                        }
                  }

                  aux[i]--; // uno menos en los de ese tipo
            }
      }
}

public static void main(String[] args)
{
      Eslabones es= new Eslabones();

      es.buscarEslabones(27, 1);
      es.mostrarResultado(27);

}

}
```

**b) (0,5 puntos) Queremos optimizar el árbol de estados desarrollados realizando podas cuando sea posible. Escribir, en lenguaje natural, alguna posibilidad de poda del árbol, especificando: si es necesario algún requisito previo, en base a qué se puede realizar y cuándo se realizaría.**

Una optimización sencilla que se puede realizar es impedir que se desarrollen nodos con más eslabones que la mejor solución que tenemos actualmente, es decir, si ya tenemos una solución con 8 eslabones, no desarrollar soluciones con 8 o más eslabones ya que por ahí no encontraremos ninguna solución mejor que la que ya tenemos y por tanto no será la óptima.
Para realizar esto simplemente añadimos a la comprobación de estado válido la comprobación `cEslab < nEslabonesOpt`.

Una segunda optimización sería no utilizar en nodos más profundos del árbol de estados eslabones que ya sabemos que podemos descartar por su longitud. Si estamos construyendo una cadena de 17 unidades de longitud, sabemos que podemos descartar el eslabón de longitud 25, ya que nunca entrará en una solución válida.
Para realizar esto tenemos al menos dos alternativas:

- Crear un array para indicar que eslabones están descartados.
- Ordenar los eslabones de mayor a menor y comenzar el bucle en la primera posición no descartada

**4. (1 punto) Se quiere resolver de una forma más eficiente el problema de los eslabones propuesto en el ejercicio 3, para ello se plantea un algoritmo voraz.**

**a) Diseñar y expresar en lenguaje natural un heurístico para este algoritmo voraz.**

Utilizar el eslabón de mayor longitud posible siempre que no exceda a la longitud que falta para completar la cadena. Para implementarlo los eslabones estarán ordenados de mayor a menor, si no se puede utilizar un eslabón, se pasa al siguiente en orden de tamaño, así hasta completar el tamaño de la cadena.

**b) Para el heurístico diseñado, razonar si es óptimo o no, y si no lo es demostrarlo.**

Este heurístico para los eslabones del ejercicio 3 (1, 12, 5, 25) **NO** es óptimo. Lo demostraremos con un contraejemplo, para ello tenemos que resolver el problema con el heurístico y con un método alternativo; pero en las dos utilizando el mismo conjunto de eslabones y ver que el número de eslabones de la forma alternativa es menor que cuando utilizamos el heurístico.
Queremos crear una cadena de longitud 15.
**Utilizando el heurístico**:
12 + 1 + 1 + 1 → 15
Se han utilizado 4 eslabones.
**Método alternativo**:
5 + 5 + 5 → 15
Se han utilizado 3 eslabones.
Con esto queda demostrado que el heurístico propuesto NO es óptimo.

**5. (1,25 puntos) El problema de la mochila 0/1 consiste en que disponemos de *n* objetos y una "mochila" para transportarlos. Cada objeto *i*= 1,2, ... *n* tiene un peso $w_i$ y un valor $v_i$. La mochila puede llevar un peso que no sobrepase *W*. Los objetos *no* se pueden fragmentar: o tomamos un objeto completo o lo dejamos. El objetivo del problema es maximizar valor de los objetos respetando la limitación de peso.**

**La función que nos proporciona la solución al problema es la siguiente:**

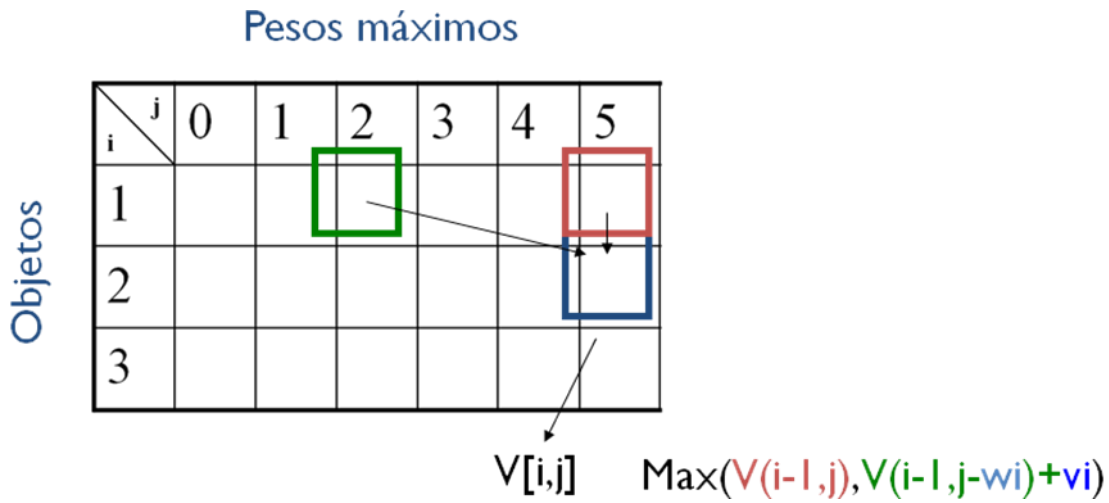
$$V(i,j) = \begin{cases} -\infty \text{ si } j < 0 \\ 0 \text{ si } i = 0 \text{ y } j \geq 0 \\ \max(V(i-1,j), V(i-1,j-w_i) + v_i) \text{ en otro caso} \end{cases}$$

**Donde i, representa a los objetos y j, representa el peso de la mochila.**
**Utilizaremos la técnica de Programación Dinámica para obtener la solución óptima al problema.**

**a) Representar la tabla necesaria para almacenar los valores intermedios (sin rellenar). Sabiendo que tenemos tres objetos y el peso máximo que puede llevar la mochila es 5.**

**b) Explicar que valores podemos rellenar de forma directa en la tabla.**

**c) Representar gráficamente sobre la tabla del apartado a) el patrón de dependencia de una celda cualquiera, es decir, marcar las celdas que son necesarias para calcular una celda dada. Se pueden plantear las hipótesis que sean necesarias sobre el valor y peso de los objetos.**




En referencia al apartado b) se representa la tabla eliminado filas y columnas de valores que se pueden calcular directamente: $j<0$, por un lado y $i=0$ y $j\geq 0$ por el otro. Por tanto, no aparece ningún valor en la tabla que se rellene de forma directa. En esta tabla estaría permitido que apareciese la fila 0 rellenada con 0 como valor directo; pero no tendría mucho sentido tener columnas negativas. La columna 0 es una columna calculada, la fórmula no proporciona valores directos para esta columna y por tanto es obligatorio que aparezca en la tabla.

**d) (0,5 puntos) Resolver este problema, escribiendo todos los valores sobre la tabla, para los siguientes objetos considerando que el peso máximo de la mochila es 5.**

| **Obj.** | **1** | **2** | **3** |
|---|---|---|---|
| **$w_i$** | **2** | **3** | **2** |
| ***$v_i$*** | **4** | **7** | **5** |

$$Max(V(i-1,j),V(i-1,j-wi)+vi)$$

| i \ j | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 4 | 4 | 4 | 4 |
| 2 | 0 | 0 | 4 | 7 | 7 | 11 |
| 3 | 0 | 0 | 5 | 7 | 9 | 12 |

| Obj. | 1 | 2 | 3 |
|---|---|---|---|
| $w_i$ | 2 | 3 | 2 |
| $v_i$ | 4 | 7 | 5 |

**6. (1,5 puntos) Cuestiones sobre la técnica de ramificación y poda (responder brevemente a lo que se plantea en cada apartado).**

**a) Se pretende resolver un problema de optimización con un algoritmo de ramificación y poda pero sólo disponemos de un heurístico de ramificación. Razonar en qué casos este algoritmo es claramente más eficiente que backtracking y en cuáles no.**

Si pretendemos buscar todas las soluciones sólo con el heurístico de ramificación y sin un algoritmo de poda que permita eliminar parte del árbol, hay que recorrerlo completo igual que haría backtracking, por tanto ramificación y poda no es más eficiente en ningún caso.
Un problema de optimización es aquel que busca un máximo o un mínimo y por tanto, implica **encontrar todas las soluciones** para quedarse con la mejor y como digo en el párrafo anterior ramificación y poda NO es más eficiente.

**b) Qué modificaciones hay que realizar en el algoritmo de recorrido en anchura para convertirlo en un recorrido en profundidad.**

Básicamente, cambiar la cola FIFO por una **pila tipo LIFO**.

Para comprobar si el nodo es solución en el mismo momento que un recorrido en profundidad clásico, habría que cambiar esta comprobación al punto donde se extraen los nodos de la pila.
Si el recorrido se quiere seguir realizando de izquierda a derecha habría que cambiar el orden en el que se meten los hijos expandidos en la pila.

**c) En el problema del puzle del cuaderno didáctico resuelto con ramificación y poda, se ven varias funciones que se aplican sobre los estados del problema. Indicar para cada una de ellos, si constituyen un heurístico o no, y si son un heurístico especificar de qué tipo:**

**a. Evaluación de estado alcanzable: El estado objetivo será alcanzable desde un estado si y sólo si Sumatorio(menor(i))+x es par. Donde x es 1 si la casilla vacía se encuentra en una de las casillas sombreadas del tablero y 0 si no es así.**

Heurístico que se aplica solamente sobre el estado inicial para comprobar que si el problema tiene solución o no. No es ni heurístico de ramificación, ni exactamente de poda, ya que como se dijo antes sólo se aplica sobre el estado inicial y sobre ninguno más, aunque podría considerarse una variante de este.

**b. Evaluación del número de fichas colocadas en la situación final del estado actual.**

Heurístico de ramificación.

**c. Sumatorio(distancia(i)) desde 1 a 16, donde distancia(i) es el número de movimientos necesarios para colocar la ficha i en la posición i.**

Heurístico de ramificación.

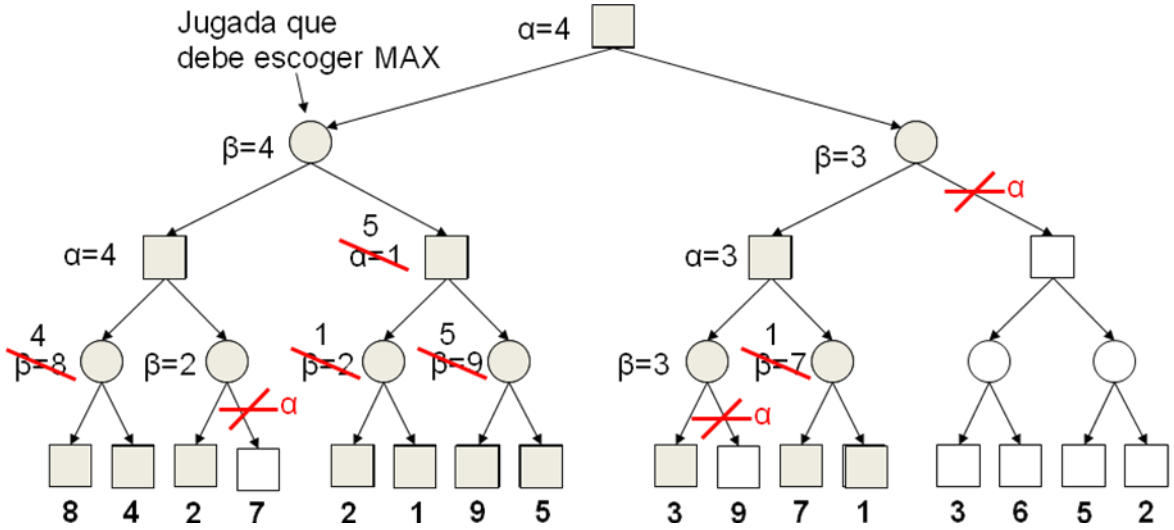
**d. Comprobación de estado repetido.**

Heurístico de poda.

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

**7. (1,25 puntos) Desarrollar la poda α-β para conocer que jugada debe realizar el jugador MAX, sobre el siguiente árbol:**




a) **Sombrear los nodos que haya que desarrollar**
b) **Escribir las cotas α y β,**
c) **Marcar los cortes e indicar si son de tipo α o β,**
d) **Por último, indicar que jugada debe elegir MAX para situarse en la mejor posición posible.**

**Notas: El jugador que realiza el primer movimiento en el árbol es MAX. Los nodos del árbol se desarrollan de izquierda a derecha.**

**Ejemplo de indicaciones:**