



Examen de Teoría de la Programación

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

Final Febrero – Curso 2006-2007

12 de febrero de 2007



DNI _____ Nombre _____ Apellidos _____
Titulación: ☐ Gestión ☐ Sistemas
Grupo al que asistes en teoría: _____

3. (1,00 puntos) El siguiente método es una implementación de la búsqueda binaria siguiendo la técnica divide y vencerás:

```
public int buscar(double x, double a[], int inicio, int fin)
{
    if (inicio <= fin)
    {
        int m = inicio + ((fin - inicio) / 2);

        if (a[m] == x)
            return m;
        else
        {
            if (x < a[m])
                return buscar(x, a, inicio, m - 1);
            else
                return buscar(x, a, m + 1, fin);
        }
    }
}
```

- a) En esta técnica, se diferencian tres pasos: descomposición del problema, resolución de los problemas elementales, combinación de los resultados para obtener la solución al problema inicial. Especificar qué operaciones del método se corresponden con cada uno de estos pasos.

Descomposición

Resolución subproblema

Combinación

- b) Cuál es la complejidad de esta solución por Divide y Vencerás. Justifica el resultado.

4. (1,25 puntos) El problema de la mochila 0/1 consiste en que disponemos de n objetos y una “mochila” para transportarlos. Cada objeto $i = 1, 2, \dots, n$ 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.

- 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 6.
- Explicar que valores podemos rellenar de forma directa en la tabla.
- Buscar un 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.
- (0,5 puntos) Escribir el código Java del método principal que implementaría la solución a este problema.

5. (1,75 puntos) Se desea formar palabras de n (o menos) letras utilizando N cubos. Para ello disponemos de un número limitado de cubos (n) con una letra diferente en cada una de sus caras y una palabra objetivo. Mediante la combinación de las caras de los cubos (sin repetir ninguno), se debe obtener la palabra objetivo. Si la combinación no fuera posible a partir de los cubos de los que se dispone, se descarta la posibilidad de formar la palabra, se informaría que es imposible hacerlo.

- (0,5 puntos) Dibujar el árbol (sólo los estados desarrollados) que desarrollará la técnica de backtracking para buscar una combinación de cubos que proporcione la palabra “mama” con las siguientes letras en los cubos.

```
private final char[] cubo1= { 'a', 'b', 'c', 'f', 'g', 'z' };
private final char[] cubo2 = { 'b', 'a', 'm', 'h', 's', 't' };
private final char[] cubo3 = { 'c', 'b', 'a', 'j', 'v', 'w' };
private final char[] cubo4 = { 'c', 'm', 'a', 'p', 'f', 'g' };
```

- (1 punto) Escribir el método que realiza el backtracking, que llamaremos *ensayar*.
- (0,25 puntos) Escribir el método *main()* que hace la llamada a *ensayar*.

(Problema expuesto en el grupo A)

6. (1,0 puntos) Se dispone de n ficheros de los que se conoce: una prioridad de valores entre 1 y 10 (correspondiendo 1 a la prioridad más baja y 10 a la más alta) y un tamaño. Se trata de maximizar la suma de las prioridades de los ficheros almacenados en un disco de una capacidad determinada, teniendo en cuenta que los ficheros no se pueden partir. Se ha de utilizar un algoritmo que calcule de manera inmediata los ficheros que se deben de incluir.

- Justificar por qué debe aplicar un algoritmo voraz para resolver este problema.
- Explicar el heurístico más adecuado para este caso.
- Demostrar si el heurístico propuesto es óptimo o no.

(Problema expuesto en el grupo C)

7. (1,0 puntos) Técnica de Ramificación y poda.

- Especifica las ventajas que puede tener el recorrido en anchura sobre Backtracking.
- Qué diferencias de la implementación implica el añadir un heurístico de ramificación al recorrido en anchura puro.
- ¿Es importante el heurístico de poda a la hora de buscar la primera solución a un problema? (Justifica la respuesta)

```

graph TD
    Root[ ] --> L1_1(( ))
    Root --> L1_2(( ))
    L1_1 --> L2_1[ ]
    L1_1 --> L2_2[ ]
    L1_2 --> L2_3[ ]
    L1_2 --> L2_4[ ]
    L2_1 --> L3_1(( ))
    L2_1 --> L3_2(( ))
    L2_2 --> L3_3(( ))
    L2_2 --> L3_4(( ))
    L2_3 --> L3_5(( ))
    L2_3 --> L3_6(( ))
    L2_4 --> L3_7(( ))
    L2_4 --> L3_8(( ))
    L3_1 --> L4_1[ ]
    L3_1 --> L4_2[ ]
    L3_2 --> L4_3[ ]
    L3_2 --> L4_4[ ]
    L3_3 --> L4_5[ ]
    L3_3 --> L4_6[ ]
    L3_4 --> L4_7[ ]
    L3_4 --> L4_8[ ]
    L3_5 --> L4_9[ ]
    L3_5 --> L4_10[ ]
    L3_6 --> L4_11[ ]
    L3_6 --> L4_12[ ]
    L3_7 --> L4_13[ ]
    L3_7 --> L4_14[ ]
    L3_8 --> L4_15[ ]
    L3_8 --> L4_16[ ]
    L4_1 --- L4_1[3]
    L4_2 --- L4_2[6]
    L4_3 --- L4_3[9]
    L4_4 --- L4_4[6]
    L4_5 --- L4_5[9]
    L4_6 --- L4_6[5]
    L4_7 --- L4_7[4]
    L4_8 --- L4_8[10]
    L4_9 --- L4_9[2]
    L4_10 --- L4_10[10]
    L4_11 --- L4_11[6]
    L4_12 --- L4_12[7]
    L4_13 --- L4_13[3]
    L4_14 --- L4_14[2]
    L4_15 --- L4_15[6]
    L4_16 --- L4_16[8]
  
```

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