

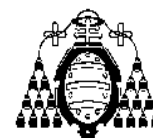


# Examen de Teoría de la Programación

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

Final Junio – Curso 2008-2009

10 de junio de 2009



DNI \_\_\_\_\_ Nombre \_\_\_\_\_ Apellidos \_\_\_\_\_  
Titulación: ☐ Gestión ☐ Sistemas

**3. (1,75 puntos) Sea un conjunto de letras de tamaño  $n$ . El problema consiste en obtener todas las palabras posibles que podamos (combinaciones de letras sin repetir ninguna) con un tamaño  $m$ , siendo  $m \leq n$ . Ejemplo: Con el conjunto de letras ABCDE queremos construir todas las palabras posibles con 3 letras, obtendríamos la siguiente respuesta: ABC, ABD... así hasta EDC. El problema se quiere resolver utilizando la técnica de Backtracking.**

- (0,5 puntos) Dibujar el árbol completo que genera la técnica del backtracking para el ejemplo propuesto.**
- (1,25 puntos) Completar el código Java para dar solución a este problema (escribirlo en los recuadros preparados a tal efecto).**

```
public class Palabras {
    private char[] arrayLetras; // conjunto de letras de tamaño n
    private int numeroLetras; // tamaño m de las combinaciones de letras
    private boolean[] estaElegida; // permite comprobar si una letra ya está elegida
    private char[] solucion; // almacena la solucion

    public Palabras(char[] arrayLetras, int numeroLetras) {
        this.arrayLetras= arrayLetras;
        this.numeroLetras= numeroLetras;
        estaElegida = new boolean[arrayLetras.length];
        solucion = new char[numeroLetras];
    }

    public void construirPalabras( ) {
        for( ) {
            if( ) {
                estaElegida[i]=true;
                solucion[posi]=arrayLetras[i];
                if( )
                    construirPalabras( );
                else{
                    imprimirArray(solucion);
                }
                estaElegida[i]=false;
            }
        }
    }

    private void imprimirArray(char[] array){ ... }

    public static void main(String[] args) {
        char[] arrayLetras= null;
        int numeroLetras= 0;

        // Lectura del conjunto de letras de tamaño n
        ...
        // Lectura del tamaño m de las combinaciones de letras
        ...
        Palabras palabras = new Palabras(arrayLetras,numeroLetras);
        palabras.construirPalabras(¿?);
    }
}
```

**4. (0,5 puntos) Sea la siguiente especificación de una función:**

- $\{Q \equiv b \neq 0\}$**
- fun *funcion*(*a*, *b*: entero) dev (*q*, *r*: entero)**
- $\{R \equiv (a=b*q+r) \wedge (r < b) \wedge (r \geq 0)\}$**

Codificar en Java el esqueleto de esta función utilizando un método público: cabecera, precondition y postcondición (no hace falta codificar la funcionalidad del método en sí).

5. (1,5 puntos) El problema de la mochila 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  $W$ . En el caso de que un objeto no se pueda meter entero, se fraccionará, quedando la mochila totalmente llena. El objetivo del problema es maximizar valor de los objetos respetando la limitación de peso. Queremos resolver este problema mediante un método voraz.

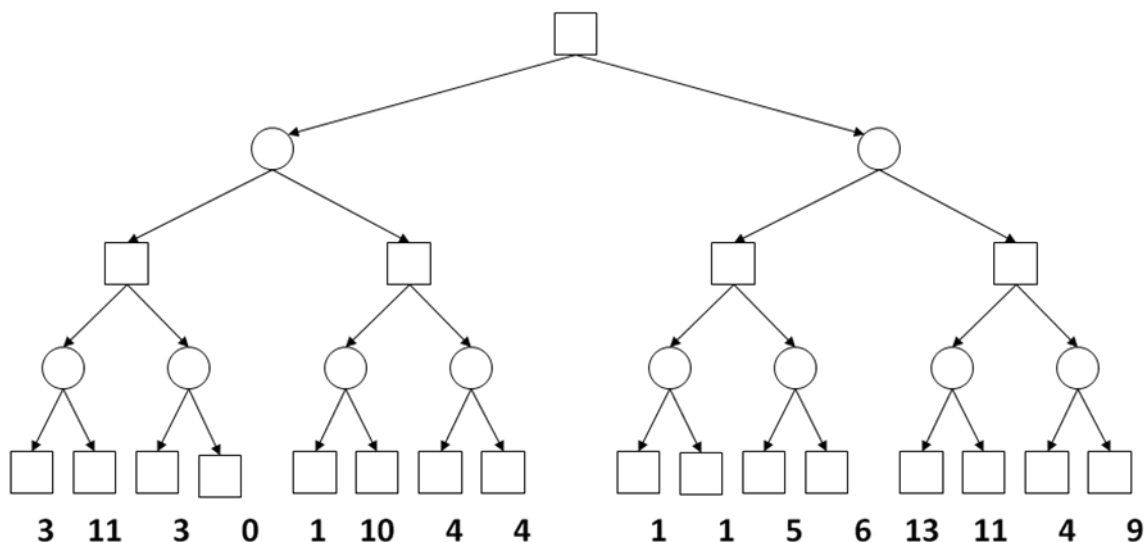
- (0,5 puntos) Cuál sería la forma de ordenar los objetos para ir metiéndolos en la mochila y alcanzar la solución óptima.
- (0,5 puntos) Cuál sería la condición de parada del bucle para ir seleccionando los objetos.
- (0,5 puntos) Qué sería lo último que meteríamos en la mochila.

6. (1,75 puntos) El problema de las  $n$  reinas consiste en colocar  $n$  reinas en un tablero de ajedrez sin que ninguna reina pueda comer a otra. Pretendemos buscar la primera solución a este problema con un tablero de  $4 \times 4$ .

Utilizaremos la técnica de ramificación y poda. El heurístico de ramificación que utilizaremos será:  $(n - n^{\circ} \text{ reinas colocadas}) * 10 + \text{diagonal}(i,j)$  para la última reina colocada, siendo  $\text{diagonal}(i,j)$  la longitud de la diagonal más larga que pasa por la casilla  $(i,j)$ .

- (0,75 puntos) Dibujar gráficamente el árbol de estados del algoritmo hasta encontrar la primera solución (sólo los estados en los que las reinas no se coman). En cada estado debemos marcar orden en el que se ha desarrollado y el valor del heurístico de ramificación.
- (0,5 puntos) Dibujar gráficamente la cola de prioridad, en el momento en que encontramos esa primera solución, representando el orden en el que quedan los estados.
- (0,25 puntos) Razonar que técnica es más eficiente para buscar la primera solución a este problema: ramificación y poda (utilizando el heurístico de ramificación dado) o backtracking.
- (0,25 puntos) ¿Y para buscar todas las soluciones posibles?

7. (1,5 puntos) Desarrollar la poda  $\alpha$ - $\beta$  para conocer que jugada debe realizar el jugador MAX, sobre el siguiente árbol:



- Sombrear los nodos que haya que desarrollar
- Escribir las cotas  $\alpha$  y  $\beta$ ,
- Marcar los cortes e indicar si son de tipo  $\alpha$  o  $\beta$ ,
- 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.