Boletín de Ejercicios Tema 3

Programación Avanzada

Grado en Ingeniería Informática y Tecnologías Virtuales

Programación III

Grado en Ingeniería del Software

Curso académico 2025/2026

Antonio M. Durán Rosal

Ejercicios Divide y Vencerás

Para cada ejercicio se pide:

a) Implementar la solución de cada ejercicio utilizando la técnica Divide y Vencerás.

b) Calcular $T(n)$ y su orden de complejidad.

Ejercicio 1: Problema posición coincidente con su valor

Dado un array unidimesional o vector ordenado de enteros v, nuestro problema es implementar un algoritmo de complejidad $O(\log n)$ capaz de encontrar un índice i tal que el contenido del vector en esa posición $v[i] = i$. Obviamente, se debe cumplir que $0 \leq i < n$. En caso de que este índice no exista, el algoritmo devolverá el valor -1.

Ejercicio 2: Problema primera posición igual

Se tienen dos vectores u y v de n enteros que cumplen la propiedad de que son distintos componente a componente hasta una posición dada, y a partir de ella, son iguales componente a componente. Es decir, si u y v son distintos hasta la componente 7, eso significa que $u[i] \neq v[i]$ para $i = 0, 1, 2, ..., 6$ y que $u[i] = v[i]$ para $i = 7, 8, ..., n - 1$. Por ejemplo, $u = [0, 2, 4, 6, 8, 10, 12, 1, 2, 3]$ y $v = [1, 3, 5, 7, 9, 11, 13, 1, 2, 3]$. Escribir un algoritmo que calcule cuál es la primera posición en la que u y v son iguales (en el caso del ejemplo, la 7) y estudiar su complejidad. El algoritmo debe poseer una complejidad menor que la lineal (que, por ejemplo, obtendríamos recorriendo ambos vectores en paralelo).

Ejercicio 3: Problema misma carga

Supongamos el siguiente contexto ficticio: en un proyecto de reconstrucción de patrimonio digital, se ha lanzado una iniciativa para escanear y preservar antiguos manuscritos mediante un sistema de escaneo paralelo. Cada escáner trabaja sobre una pila de páginas históricas, digitalizándolas una a una. Para asegurar que el escaneo se produce de manera eficiente y equilibrada, se han organizado turnos de técnicos especializados en grupos consecutivos.

Se ha descubierto que el proceso de escaneo es óptimo si la carga de trabajo (medida en minutos estimados para procesar cada página asignada) está equilibrada entre la primera y la segunda mitad de cada grupo de técnicos. Además, esta condición de equilibrio debe cumplirse de forma recursiva para cada subgrupo.

Por ejemplo:

Luis, 2 - Marta, 4 - Sergio, 1 - Clara, 5 - Raúl, 3 - Elena, 3 - Tomás, 4 - Alicia, 2

Para verificar qué grupos de técnicos cumplen esta propiedad se utilizará una función llamada **mismaCarga**.

Se pide:

a) Defina la clase **Tecnico**, que constará de dos atributos: **nombre** y **carga** (minutos estimados). Incluya constructor con dos parámetros, métodos observadores y modificadores, sobrecarga del operador $=$ y una función de salida `show()` que muestre el objeto en formato compacto: `(Nombre, carga)`. No es necesario incluir ninguna otra funcionalidad.

b) Implemente la función **mismaCarga**, que reciba un vector de `Tecnico` y devuelva la suma total de carga si se cumple la propiedad descrita, o -1 en caso contrario. Se considera caso base cuando el tamaño del vector es 2, en cuyo caso se devuelve directamente la suma de las cargas.

Ejercicio 4: Problema equiproducto

Dada una matriz de enteros mayores o iguales a cero de dimensión NxN con N potencia de 2, se dice que esa matriz es "equiproducto" si los elementos de las 4 submatrices que la componen producen el mismo producto y a su vez cada submatriz es también equiproducto. Además, una matriz de 2x2 siempre se considera equiproducto. La siguiente matriz es equiproducto ya que el producto de sus cuatro submatrices es 168 y además esas submatrices son de tamaño 2x2 y por tanto también lo son. Observe también que para el citado ejemplo el producto de todos sus elementos es 796594176.

1	3	2	14
7	8	3	2
4	6	2	2
7	1	21	2

Ejercicio 5: Problema mínimo en terreno

Disponemos de una matriz m (cuadrada y potencia de dos) que representa la digitalización de un terreno que queremos analizar. El valor $m[x][y]$ representará la altura real del terreno en la posición (x, y). Queremos buscar en la matriz m el punto más bajo del terreno. Dada las características del terreno, sabemos que en la matriz sólo existe un punto con estas características. También dispondremos de la función minimo(p1, p2, p3, p4) de complejidad constante que devuelve el mínimo de cuatro valores.

Se pide resolver el problema usando la técnica Divide y Vencerás. Esta función devolverá un valor flotante que represente la altura más baja del terreno. En esta ocasión la llamada a la función tendrá en cuenta los valores x_1, y_1 (esquina superior izquierda del terreno), y x_2, y_2 (esquina inferior derecha). El caso base se produce cuando sólo tenemos una matriz de un elemento, el cuál será directamente el mínimo.

Ejercicio 6: Problema del filtro

El departamento de software de una empresa ha recibido el encargo de realizar un filtro que permita saber si una imagen cumple una determinada propiedad. La imagen se recibe en forma de una matriz m cuadrada de nxn números enteros, siendo n una potencia de 2. Cada número de dicha matriz representa un pixel de la imagen, que puede tomar valores desde 0 a 255. El objetivo del encargo es diseñar un algoritmo por la técnica de divide y vencerás que dada una matriz inicial, devuelva verdadero o falso si dicha matriz cumple un filtro: **boolean filtro (int ** m, int n)**.

La imagen cumple el filtro si $a < b$, siendo a el valor guardado en el pixel superior izquierdo de la matriz, y b el valor guardado en el pixel inferior derecho de la matriz; y también se debe cumplir recursivamente, en cada una de las cuatro submatrices que la componen, que $a < b$, hasta llegar a submatrices tengan 2x2 puntos. Ejemplo: La imagen de 4x4 puntos que se muestra a la izquierda cumple el filtro ya que $1 < 84$; y a su vez en las cuatro submatrices que se obtendrían se cumple también el filtro, ya que $1 < 8$ y $10 < 19$ y $20 < 43$ y $55 < 84$. No es necesario dividir más ya que las submatrices son 2x2. Por tanto la imagen pasa el filtro.

1	2	10	14
5	8	15	19
20	25	55	70
40	43	71	84

Matriz 4x4

1	2	10	14	20	25	55	70
5	8	15	19	40	43	71	84

Matrices 2x2

Se pide implementar la función **filtro** que devolverá true si se cumple el filtro.

Ejercicios Voraz y Backtracking

Para cada ejercicio se pide:

a) Completar las clases que representan los objetos del problema, la solución, el estado y el problema en sí.

b) Completar los algoritmos Voraz y Backtracking.

Ejercicio 7: Problema de asignación de tareas

Dados n agentes y n tareas, existe un coste $C(i, j)$ si el agente i ejecuta la tarea j. Se pide buscar una asignación de tareas a personas para que el coste sea mínimo. Cada tarea es asignada a una persona y cada persona tiene una sola tarea. De ahí que haya n agentes y n tareas. La solución buscada es el par Persona-Tarea asignada. Las clases del problema serán Tarea, Agente, SolucionAsignacion, ProblemaAsignacion y EstadoAsignacion.

Ejercicio 8: Problema recubrimiento de conjuntos

Dado un conjunto U de elementos e_i, $i \in [0, n-1]$ y un conjunto S de m conjuntos s_j, cuya unión es igual al universo, y un peso $w_i \geq 0$ asociado a cada conjunto, el problema es identificar el subconjunto con menor coste de S cuya unión es igual al universo U.

Por ejemplo, considere el universo $U = \{1, 2, 3, 4, 5\}$ y el conjunto $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$, todos ellos con peso 1, podemos comprobar que la unión de todos los conjuntos en S es U y que podemos cubrir la totalidad de los elementos con los conjuntos $S = \{\{1, 2, 3\}, \{4, 5\}\}$.

Las clases serán Subconjunto, SolucionRecubrimiento, ProblemaRecubrimiento y EstadoRecubrimiento.

Ejercicio 9: Problema tareas y procesadores

Dada una lista de m tareas con duraciones d_j y un conjunto de n procesadores buscar la asignación de tareas a procesadores tal que el tiempo total de ejecución sea mínimo. Las clases serán Tarea, Procesador, SolucionTareas, ProblemaTareas y EstadoTareas.

Ejercicio 10: Problema suma de subconjuntos

Tenemos n números positivos distintos (usualmente llamados pesos). El objetivo es encontrar todas las combinaciones de estos números que sumen M. En este caso, las clases serán SolucionSubconjuntos, ProblemaSubconjuntos y EstadoSubconjuntos.

Ejercicio Entregable

Se pide:

a) Definir las clases que representan los objetos del problema, la solución, el estado y el problema en sí.

b) Definir los algoritmos Voraz y Backtracking.

Ejercicio 11: Problema personalizado (10)

Resolver un problema desde cero que entregará el profesor de forma personalizada y presentarlo en clase.

La evaluación de esta parte se realizará aplicando la fórmula:

$$Nota = 60\,\% \cdot Profesor + HC(20\,\%Coevaluacion) + HA(20\,\%Autoevaluacion)$$

siendo HC la ponderación de la Heteroevaluación con respecto a la evaluación de vuestros compañeros, y HA con respecto a vosotros mismos. Será 1 si la diferencia de vuestra nota con respecto a la mía es menor o igual a 1 punto, 0.5 si la diferencia está en el intervalo $(1, 2.5]$, o 0 si es mayor a 2.5.

Para la evaluación, se tendrá que considerar:

- La solución al problema.
- La explicación del grupo.
- La respuesta dadas al profesor.
- Imprescindible que funcione.

Opción 1: Puzzle de Colores (máx 8)

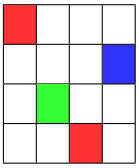
Se dispone de un tablero de $N \times N$ casillas que debe rellenarse utilizando una paleta de k colores distintos. El objetivo es colorear completamente el tablero cumpliendo un conjunto de restricciones locales y globales.

Algunas casillas pueden venir ya coloreadas de inicio y no podrán modificarse. El resto de casillas deberán ser rellenadas por el programa.

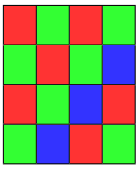
Las restricciones que debe cumplir una solución válida son las siguientes:

1. **Restricción de adyacencia:** dos casillas ortogonalmente adyacentes (arriba, abajo, izquierda, derecha) no pueden tener el mismo color.
2. **Restricción global de uso de colores:** para cada color i en $[1 \dots k]$ se especifica una condición de uso global, que puede ser:
   - un número *fijo* de apariciones (por ejemplo, que el color i aparezca exactamente T_i veces en todo el tablero), o
   - un *intervalo* de apariciones permitido $[m_i, M_i]$, indicando el mínimo y el máximo de veces que el color i puede aparecer.
3. **Casillas prefijadas:** algunas posiciones del tablero pueden venir ya coloreadas. El color de estas casillas no se puede cambiar y debe respetarse en todo momento.

Puzzle inicial



Solución



En la figura se muestra un ejemplo de tablero con algunas casillas ya coloreadas. El objetivo es rellenar el resto de casillas respetando todas las restricciones sabiendo que

tenemos 3 colores, que el color azul debe aparecer 3 veces, que el color rojo debe aparecer entre 3 y 6 veces, y que el color verde debe aparecer entre 4 y 8 veces.

Consideraciones:

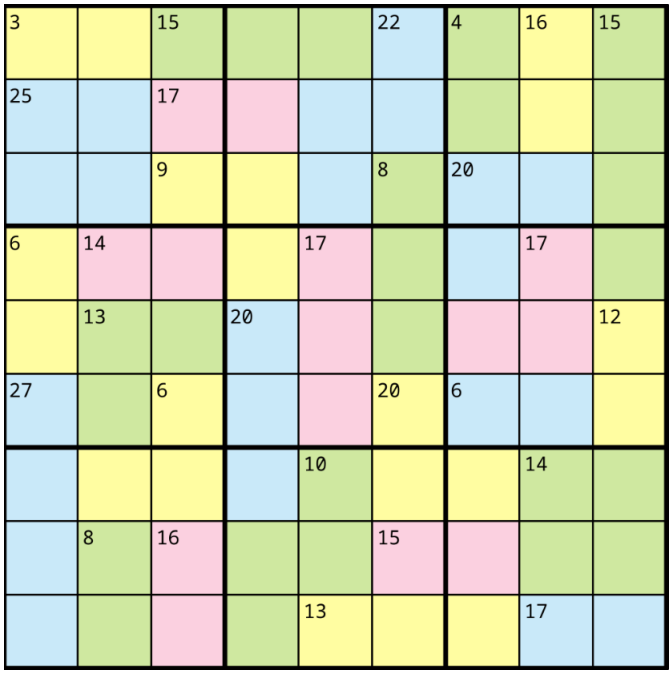
- El programa debe permitir introducir:
  - el tamaño del tablero N y el número de colores k,
  - las casillas inicialmente coloreadas,
  - para cada color i, o bien un número fijo de apariciones T_i, o bien un intervalo permitido $[m_i, M_i]$.
- El problema debe resolverse utilizando exclusivamente **Backtracking**.
- Se deben mostrar todas las soluciones posibles o indicar que no existe ninguna solución para los datos de entrada proporcionados.
- Se valorará especialmente la representación visual del tablero (por ejemplo, utilizando colores, símbolos o una interfaz gráfica) y la claridad en la visualización de la solución final.

Opción 2: Killer Sudoku (máx 9)

El Killer Sudoku es una variación del Sudoku clásico en el que, además de las reglas habituales de filas, columnas y subcuadros, el tablero contiene *jaulas* (o regiones irregulares) que imponen una restricción adicional: la suma de los valores en cada jaula debe coincidir exactamente con un número objetivo. Cada jaula puede estar formada por un número arbitrario de casillas, contiguas entre sí, y se marca indicando la suma total que deben alcanzar.

En este ejercicio trabajaremos con un tablero de 9×9 dividido en jaulas de distintos tamaños. Se deben cumplir simultáneamente las siguientes restricciones:

1. En cada fila deben aparecer los enteros del 1 al 9 sin repetirse.
2. En cada columna deben aparecer los enteros del 1 al 9 sin repetirse.
3. En cada subcuadro de 3×3 deben aparecer los enteros del 1 al 9 sin repetirse.
4. En cada jaula, la suma de los valores asignados a sus casillas debe coincidir con el valor objetivo indicado.
5. Dentro de una misma jaula no puede repetirse ninguna cifra, independientemente de que las posiciones pertenezcan o no a la misma fila, columna o subcuadro.



En la figura se observa un ejemplo de tablero con jaulas. Cada jaula se representa con un borde irregular y un número que indica la suma total que deben alcanzar las casillas que contiene. La solución sería la siguiente:

Consideraciones:

- El programa debe permitir introducir:
  - la estructura de jaulas (cada una definida por sus coordenadas y su suma objetivo),
  - las posiciones iniciales que ya están fijadas, si las hubiera.
- El algoritmo debe resolver el tablero usando exclusivamente **Backtracking**.
- Se deben mostrar todas las soluciones posibles, o indicar que no existe solución.
- Se valorará la representación visual del tablero y las jaulas, así como la visualización de la solución final.

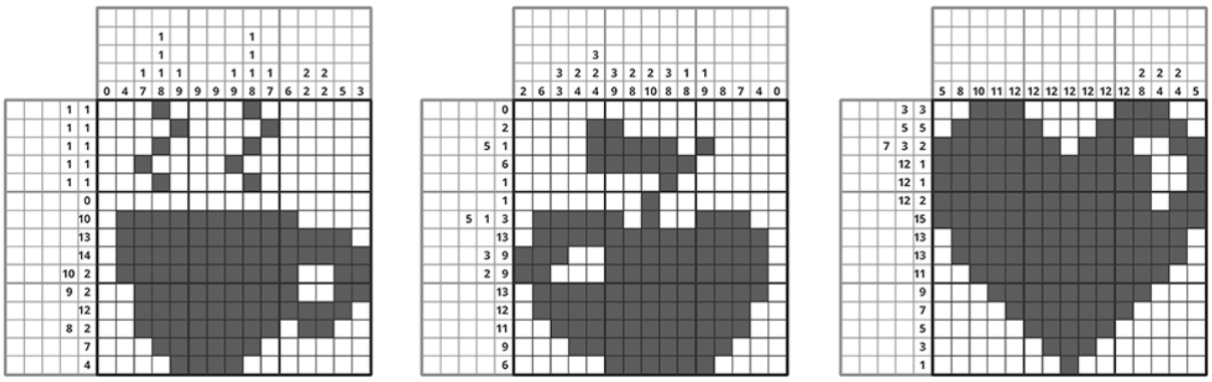
Opción 3: Nonograma (máx 10)

Un nonograma (o *picross*) es un rompecabezas de lógica en el que se debe determinar qué casillas de un tablero deben rellenarse (por ejemplo, en negro) y cuáles deben quedar vacías, de forma que se cumplan unas pistas numéricas asociadas a cada fila y a cada columna.

El tablero se modelará como una matriz de tamaño $N \times M$ donde cada posición puede estar en uno de dos estados: casilla rellena o casilla vacía.

Para cada fila y para cada columna se proporciona una secuencia de enteros que describe los tamaños de los bloques consecutivos de casillas rellenas que deben aparecer en esa fila o columna, en orden.

Por ejemplo, si la pista de una fila es `[3 1]`, significa que en esa fila debe haber un bloque de 3 casillas consecutivas rellenas, seguido (tras al menos una casilla vacía) de un bloque de 1 casilla rellena. Las casillas vacías restantes pueden situarse al principio, entre bloques o al final, siempre respetando el orden de los bloques indicados.



En la figura se muestran tres ejemplos de nonogramas con sus pistas de filas y columnas y una posible solución, donde las casillas rellenas forman una imagen sencilla.

Consideraciones:

- El programa debe permitir introducir:
  - el tamaño del tablero (N filas y M columnas),

    - las pistas de cada fila,
    - las pistas de cada columna,
    - opcionalmente, algunas casillas prefijadas como rellenas o vacías.
- El problema debe resolverse utilizando exclusivamente **Backtracking**.
- Se deben mostrar todas las soluciones posibles o, en su defecto, indicar que no existe ninguna solución para las pistas introducidas.
- Se valorará la representación visual del tablero resultante (por ejemplo, mostrando las casillas rellenas y vacías de forma clara, o incluso generando una pequeña "imagen" a partir de la solución).