Convocatoria ordinaria – 18 de mayo de 2023 (Parte 2)

Apellidos, nombre _____ NIF: _

Pregunta 1 (2,5 p.)

Estamos jugando un juego contra una computadora. La idea es distribuir nuestras N bolas rojas en 8 grupos de diferentes tamaños. La computadora hará lo mismo con sus M bolas azules. Es decir, creará otros 8 grupos de diferentes tamaños. Debemos asignar a cada uno de nuestros grupos de color rojo un grupo de color azul. Ganaremos un punto cada vez que el número de bolas rojas sea mayor que el número de bolas azules en una tarea. Diseñe lo siguiente:

a) (1 p.) Un algoritmo codicioso para calcular la solución óptima (maximizar el número total de puntos obtenidos). La complejidad temporal del algoritmo debe ser lo más baja posible.

b) (0,5 p.) ¿Cuál es la complejidad del algoritmo diseñado?

c) (1 p.) Haz una traza del algoritmo para el siguiente caso (n=8) para permitirnos maximizar el número de puntos. Después de eso, ¿cuántos puntos conseguimos contra el ordenador?

- Tenemos los siguientes tamaños para los grupos rojos = 9, 48, 45, 10, 42, 43, 19, 28
- Tenemos los siguientes tamaños para los grupos azules = 33, 25, 69, 29, 76, 42, 23, 4

Pregunta 2 (2,5 p.)

Dado un conjunto de números enteros positivos, se pide encontrar si es posible seleccionar un subconjunto de elementos cuya suma sea igual a un valor objetivo dado.

Formalmente, se da un conjunto de números enteros positivos $\{a_1, a_2, ..., a_n\}$ y un valor objetivo S. Se debe determinar si existe un subconjunto de estos números cuya suma sea igual a S.

Por ejemplo, considere el siguiente conjunto de números: {3, 7, 2, 9} y el valor objetivo S = 13. En este caso, es posible seleccionar los números 3, 7 y 2 para obtener una suma de 12, que es menor que S. Sin embargo, no hay forma de obtener una suma exacta de 13, por lo que la respuesta sería False.

a) (1,5 p.) Implementar una solución eficiente en Java utilizando programación dinámica. La signatura del método será la siguiente:

```
public static boolean subsetSum(int[] numbers, int target)
```

b) (1 p.) Representa los valores intermedios que genera el algoritmo para el caso descrito anteriormente números= {3, 7, 2, 9} para obtener un objetivo de S= 13

Pregunta 3 (2,5 p.)

El Sudoku es un rompecabezas de colocación de números basado en la lógica. En el Sudoku clásico, el objetivo es llenar una cuadrícula de 9 × 9 con dígitos para que cada columna, cada fila y cada una de las nueve subcuadrículas de 3 × 3 que componen la cuadrícula (también llamadas "cajas", "bloques" o " regiones") contienen todos los dígitos del 1 al 9. El colocador de rompecabezas proporciona una cuadrícula parcialmente completa, que para un rompecabezas bien planteado tiene una única solución.

Por ejemplo, el siguiente podría ser el estado inicial de un tablero, con estos números que permanecerán fijos y a la derecha una posible solución que cumple con las restricciones (cada columna, cada fila y cada una de las nueve subcuadrículas de 3 × 3 que componen la cuadrícula contienen todos los dígitos del 1 al 9):

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Completa el código a continuación para resolver este problema utilizando un algoritmo de backtracking.

```
public class Sudoku {
    public final int N = 9; //size of the Sudoku
    public int[][] board; //board for the Sudoku puzzle
    public boolean wasFound; //whether a solution was found (true or false)

    public static void main(String arg[]) {
        Sudoku sudoku = new Sudoku();
        
        if (!sudoku.wasFound) System.out.println("THERE IS NO SOLUTION");
    }

    public Sudoku() {
        String name = "FILE.txt"; //name of the file
        board = new int[N][N];
        readFile(name);
        System.out.println("THE SUDOKU TO BE SOLVED IS:");
        writeBoard();
    }

    //method to load the initial state of the game in “board”. It is already
implemented.
    private void readFile(String name) { }

    //method to print the “board”. It is already implemented.
    private void writeBoard() { }

    public void backtracking() {
        wasFound = false;
        int[]zero = new int[2]; // almacena fila y columna del siguiente hueco
        zero = findZero(0, -1); // busca el siguiente hueco comenzando en fila 0,
columna -1
        backtracking(                );
    }

    private void backtracking(int i, int j) {
        if (                    ) { //we found a solution (AND FINISH)
            wasFound = true;
            System.out.println("SOLUTION FOUND:");
            writeBoard();
        }
        else
            for (                    ) {
                if(          && row(i,k) && column(j,k) && region(i,j,k)) {

                    
                    int[]zero = new int[2];
                    
                    backtracking(zero[0], zero[1]);
                    

              }
           }
    }

    //look for the next position with 0 (ie., that it is empty)
    private int[] findZero(int i, int j) {…}// devuelve [n, n] si no quedan huecos

    //to check if the row i is valid for the number k
    private boolean row(int i, int k) {…}
```

```
    //to check if the column j is valid for the number k
    private boolean column(int j, int k) {    }

    //to check if the region is valid for the number k
    private boolean region(int i, int j, int k) {
        boolean b = true;
        int startingI = i-i%3; //if for example i=7 => startingI = 6
        int startingJ = j-j%3; //if for example j=1 => startingJ = 0
        for (                                                    )
            for (                                                    )
                if (board[posI][posJ] == k) b=false;
        return b;
    }
}
```

Pregunta 4 (2,5 p.)

Contesta de forma breve (tres líneas como máximo por respuesta). Sea el problema de la asignación de tareas a agentes visto en clase.

a) Si tenemos 6 tareas y 6 agentes. Al resolver el problema con backtracking ¿Cuántos estados hijos tendrá el estado inicial? ¿Qué altura tendrá el árbol de estados?
b) Para este mismo caso ¿cuál sería el número de total de estados generado?
c) ¿Qué complejidad tendría este algoritmo?
d) ¿Qué diferencia tendrá el árbol desarrollado con backtracking y con ramifica y poda?
e) ¿Qué estados hijos tiene el estado (A-3, B-5, C-1)
f) ¿Qué mecanismo utiliza Ramifica y Poda para seleccionar el siguiente nodo a desarrollar?
g) ¿Qué significado tiene el valor del heurístico de ramificación en un nodo hoja?
h) ¿Qué ocurre en el desarrollo del árbol si el heurístico de ramificación devuelve el mismo valor para todos los estados?
i) ¿Si el heurístico de ramificación tiene poca calidad (como por ejemplo, únicamente contar el número de tareas que faltan por asignar), llegaríamos a encontrar una solución óptima a un problema?
j) ¿Qué misión tiene el heurístico de poda en un problema en el que buscamos la primera solución válida?