UNIVERSIDAD DE MÁLAGA

Análisis y Diseño de Algoritmos
Parcial 5 (Diciembre 2025)

Nombre: **Grupo:**

1.(10 puntos) Parte teórica.

Dado un tablero cuadrado T de dimensión $n \times n$, una casilla de inicio *(xIni, yIni)* y una casilla de fin *(xFin, yFin)*, queremos encontrar el camino más corto que un caballo de ajedrez puede realizar para ir de la posición de inicio a la posición de fin. En el tablero hay algunas casillas que representan fosos muy profundos, por lo que el caballo no podrá colocarse sobre ellas (aunque sí saltar sobre ellas).

Queremos **diseñar** un algoritmo de **Vuelta Atrás** que resuelva el problema y nos indique **en una matriz** las **casillas por las que va pasando el caballo y el orden en el que lo va haciendo**. El algoritmo debe trabajar con los siguientes datos de entrada:

- Matriz de números enteros **T** de dimensión ***n x n***, donde **T(i,j) = -1** si hay un foso y **T(i,j) = 0** si no lo hay.
- Casilla de inicio ***(xIni, yIni)*** , donde $0 \leq xIni, yIni < n \wedge T(xIni, yIni) = 0$.
- Casilla de fin ***(xFin, yFin)*** , donde $0 \leq xFin, yFin < n$.

a) (10 ptos.) Diseñar el algoritmo de Vuelta Atrás para resolver el problema. Se debe indicar claramente la estructura de la solución, el estado inicial de la misma, la función de terminación, la función objetivo y la política de ramificación.

Estructura de la solución: Matriz de enteros *S* de *n x n*, donde S(i,j)=-1 si en la casilla hay un foso, S(i,j) = 0 si la casilla no ha sido utilizada por el caballo y S(i,j)= k > 0 si el caballo estuvo en esa casilla en el k-ésimo paso de construcción (consideramos que el estado inicial de la solución es el paso de construcción 1)
Para realizar una implementación eficiente sería muy útil conocer también la posición actual del caballo (x_a, y_a). Por tanto, podemos considerar la solución como una terna (S, x_a, y_a) que contendrá la matriz y la posición actual del caballo.

Estado inicial de la solución: la matriz solución será una copia del tablero dado (o el mismo tablero si se puede modificar) donde hemos colocado un 1 en la casilla de inicio. Además, la posición actual del caballo será la casilla de inicio.
Es decir, tendremos una tupla *(S, xIni, yIni)* donde S se define como

$$S(i,j) = \begin{cases} T(i,j) & (i,j) \neq (xIni, yIni) \\ 1 & (i,j) = (xIni, yIni) \end{cases}$$

Política de ramificación: dada una solución no completa (S, x_a, y_a), tenemos que indicar las posibles opciones válidas que tenemos para modificarla y hacerla un poco más completa. En nuestro caso, intentaremos que el caballo avance un paso más. Por ello, las restricciones explícitas definen las opciones a priori que tenemos son los 8 posibles saltos del caballo desde la casilla actual

$$candidatos = \{ (x_a + 1, y_a + 2), (x_a + 2, y_a + 1), (x_a + 2, y_a - 1), (x_a + 1, y_a - 2), (x_a - 1, y_a - 2), (x_a - 2, y_a - 1), (x_a - 2, y_a + 1), (x_a - 1, y_a + 2) \}$$

Por otro lado, las restricciones implícitas indican cuando un valor candidato es válido para modificar el estado de la solución actual. En nuestro caso, la siguiente posición del caballo será válida si es una posición que no se sale del tablero, que no tiene un foso y por el que no hemos pasado (en otro caso entraríamos en bucle infinito).

Es decir, sea $(x_c y_c) \in candidatos$ debe cumplir $0 \leq x_c < n \wedge 0 \leq y_c < n \wedge S(x_c, y_c) = 0$

Función de terminación: tendremos una solución completa cuando el caballo haya llegado a la casilla final. Es decir, si (S, x_a, y_a) es la solución, entonces hay que comprobar $(x_a, y_a) = (xFin, yFin)$

Función objetivo: queremos el camino más corto, por lo que dada una solución completa (S, x_a, y_a), tendremos que ver el número de pasos que se utilizó en el recorrido. Como incrementamos el número que asignamos a la matriz cada vez que damos un paso, no tenemos más que ver el último número que hemos añadido a S. Es decir,
calidad$((S, x_a, y_a)) = S(x_a, y_a)$

2. (*10 puntos*) Parte práctica.

Implementa el algoritmo de Vuelta Atrás diseñado en el ejercicio anterior.

```
public static int[][] encontrarCamino(int[][] tablero, int xIni,
                                      int yIni, int xFin, int yFin) {
     int[][] sol = copiar(tablero);
     sol[xIni][yIni] = 1;
     return encontrarCamino_VA(sol, xIni, yIni, xFin, yFin, null);
}

private static int[][] encontrarCamino_VA(int[][] sol, int xActual,
                   int yActual, int xFin, int yFin, int[][] mejor) {

     if (xActual == xFin && yActual == yFin) {
          if (calidad(sol, xFin, yFin) <
               calidad(mejor, xFin, yFin)) {
               mejor = copiar(sol);
          }
     } else {
          int dir = 1;
          while (dir <= 8) {
               int xSig = sigX(xActual, dir);
               int ySig = sigY(yActual, dir);
               if (valida(xSig, ySig, sol)) {
                    sol[xSig][ySig] = 1 + sol[xActual][yActual];
                    mejor = encontrarCamino_VA(sol, xSig, ySig,
                                               xFin, yFin, mejor);
                    sol[xSig][ySig] = 0;
               }
               dir++;
          } // while
```

```
        } // if-else
        return mejor;
}

private static boolean valida(int x, int y, int[][] sol) {
        int n = sol.length;
        return (x>=0 && x < n && y>=0 && y< n && sol[x][y]==0);
}

private static int calidad(int[][] sol, int x, int y) {
        return (sol == null) ? Integer.MAX_VALUE : sol[x][y];
}

private static int sigX(int x, int dir) {
        int sig = 0;
        switch (dir) {
        case 1:
        case 8:
                sig = x - 2;
                break;
        case 2:
        case 7:
                sig = x - 1;
                break;
        case 3:
        case 6:
                sig = x + 1;
                break;
        default:
                sig = x + 2;
        }
        return sig;
}

private static int sigY(int y, int dir) {
        int sig = 0;
        switch (dir) {
        case 2:
        case 3:
                sig = y + 2;
                break;
        case 1:
        case 4:
                sig = y + 1;
                break;
        case 6:
        case 7:
                sig = y - 2;
                break;
        default:
```

```
            sig = y - 1;
        }
        return sig;
}
```