Reto 2. Abstracción

Manuel Bolaños Quesada

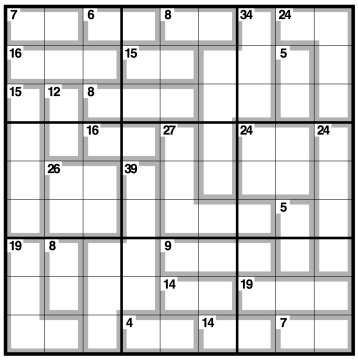
Índice

1. Introducción

El objetivo es especificar el T.D.A. Sudoku Killer, con los métodos de los que se dota la clase, y otras posibles clases auxiliares para poder así resolver un Sudoku Killer. A continuación, muestro los tipos de datos abstractos que he utilizado para resolver un Sudoku clásico, y con una leve modificación también puede resolver un Sudoku Killer.

2. Tipos de datos abstractos utilizados

Un Sudoku Killer está compuesto, principalmente, por un tablero en el que se guardan los números del sudoku. Además, hay varias rejas"que son una pequeña colección de casillas adyacentes, que tienen que sumar un determinado número.



Así pues, para hacer el T.D.A. Sudoku, he utilizado tres clases: Casilla, Reja y Sudoku. Se espefican más detalladamente a continuación:

2.1. Clase Casilla

```
/**
* @brief T.D.A. Casilla {
*
* Una instancia del tipo de dato Casilla nos permite almacenar unas coordenadas
* correspondientes a una casilla del tablero de un Sudoku.
*
* Proporciona además métodos para consultar las coordenadas que forman esta casilla
*/
class Casilla {
private:
    /**
    * @brief entero que guarda la coordenada x de la casilla
    */
    int x;

    /**
    * @brief entero que guarda la coordenada y de la casilla
    */
    int y;
public:
    /**
     * @brief Inicializa una casilla a la casilla default (0, 0)
     */
    Casilla();
```

```
        /**
         * @brief Inicializa una casilla a la casilla (x, y)
         * @param x Coordenada x de la casilla
         * @param y Coorrdenada y de la casilla
         */
        Casilla(int x, int y);

        /**
         * @brief Coordenada x de la casilla
         * @return La coordenada x de la casilla
         */
        int getX() const;

        /**
         * @brief Coordenada y de la casilla
         * @return La coordenada y de la casilla
         */
        int getY() const;
};
```

2.2. Clase Reja

```
/**
* @brief T.D.A. Reja
*
* Una instancia del tipo de dato Reja nos permite almacenar una de las varias rejas que
* tiene un sudoku killer
*
* Proporciona también métodos para obtener la infomración que guardan
*/
class Reja {
private:
        /**
         * @brief Entero que guarda la suma de las casilla que forman la reja
         */
        int suma;

        /**
         * @brief Numero de casillas de la reja
         */
        int num_casillas;

        /**
         * @brief Puntero a las casillas que forman la reja
         *
         * El tamaño de @p casillas debe ser @p num_casillas
         */
        Casilla *casillas;

public:
        /**
                * @brief Inicializa una reja a los valores predeterminados
         *
         * @post todos los valores se ponen a 0
         */
```

```
        Reja();

        /**
         * Inicializa una reja
         * @param suma Suma de las casillas de la reja
         * @param num_casillas Numero de casillas que tiene la reja
         * @param casillas Casillas que forman la reja
         * @pre suma ¿ 0
         * @pre num_casillas ¿ 0
         */
        Reja(int suma, int num_casillas, int *casillas);

         /**
         * @brief Suma de las casillas de la reja
         * @return La suma de las casillas de la reja
         */
        int getSuma() const;

        /**
        * @brief Numero de casillas de la reja
        * @return El numero de casillas de la reja
        */
        int getNumCasillas() const;

        /**
        * @brief Consulta la casilla en una determinada posicion
        * @param i posicion de la que se quiere saber la casilla
        * @pre 0 <= i < num_casillas
        * @return La casilla correspondiente a la posicion i
        */
        Casilla getCasillaAt(int i) const;
};
```

2.3. Clase Sudoku

```
/**
* @brief T.D.A. Sudoku
*
* Una instancia del tipo Sudoku es, en esencia, un tablero cuadrado con 81 casillas
* en el que cada casilla contiene un numero, y sige las reglas básicas de un sudoku
*
* El sudoku se puede convertir en sudoku killer tan solo con poner como "true" una variable
* privada
*
* Proporciona métodos para la resolución del juego, así como métodos de consulta, modificación

* y para mostrar el tablero
*/
class Sudoku {
private:
        /**
         * @brief Matriz de 9x9 que contiene los números del tablero del sudoku
         */
        int tablero[9][9];
```

```
    /**
     * @brief Variable que indica si el sudoku es killer o no
     */
    bool killer;

    /**
     * @brief Numero de rejas que tiene el tablero (en caso de ser killer)
     */
    int num_rejas;

    /**
     * Array de rejas, que apunta a las distinitas rejas que conforman el tablero (en caso de
     * ser killer)
     */
    Reja *rejas;
public:
    /**
     * @brief Inicializa el sudoku
     *
     * @post Rellena el tablero con ceros, killer = false
     */
    Sudoku();

    /**
     * @brief Inicializa el sudoku
     * @param tablero Tablero que se usa para el juego
     *
     * @post El tablero del juego es el pasado por parámetros, y killer = false
     */
    Sudoku(int tablero[][9]);

    /**
    * @brief Inicializa el sudoku
     * @param tablero Tablero que se usa para el juego
     * @param num_rejas Numero de rejas que tiene el sudoku (killer)
     * @param rejas array que apunta a las rejas que conforman el tablero
     *
     * @post killer = true
     */
    Sudoku(int tablero[][9], int num_rejas, Reja *rejas);

    /**
     * @brief Numero que hay en una posicion determinada del tablero
     * @param posicion Posicion en la que consultar el numero
     * @pre 0 <= posicion < 81
    * @return El numero contenido en la posicion @p posicion del tablero
     */
    int getNumeroAt(int posicion) const;

    /**
     * @brief Encuentra la siguiente casilla vacía a una dada (@p otra)
     * @param otra La casilla de la que hallar la siguiente casilla vacía
     * @return La siguiente casilla vacía
     * @post No se modifica nada
     */
```

```
    Casilla siguienteCasillaVacia(Casilla & otra) const;

    /**
     * @brief Compprueba si un valor dado es válido en un casilla dada, con las reglas del
     * sudoku (killer)
     * @param casilla La casilla en la que comprobar si es válido el número
     * @param numero El numero a comprobar válido en la casilla dada
     * @return true si es válido, y false si no lo es
     */
    bool esValido(Casilla casilla, int numero) const;

    /**
     * @brief Soluciona el juego de manera recursiva:
     *
     * El método recibe un casilla, y a partir de ella, coloca números del 1 al 9 en las
     * siguiente casillas vacías del tablero.
     * Empieza colocando el 1, y comprueba si es válido, si no lo es, coloca un 2, y así
     * sucesivamente. Si algún número es válido, se vuelve a llamar al método (recursividad)

     * y hace lo mismo para la siguiente casilla vacía. Si llegado a un punto ningún número
     * del 1 al 9 es válido, vuelve atrás hasta el último número válido colocado, lo elimina, y
     * sigue probando con números mayores.
     * @param casilla Casilla a partir de la que coloca los números. Por defecto, será la
     * casilla de la esquina superior izquierda
     * @return true si se ha resuelto con éxito, y false en otro caso
     */
    bool solucionar(Casilla casilla = Casilla(0, 0));

    /**
     * @brief Consulta la reja que hay en una determinada posición
     * @param num Posición en la que consultar la reja
     * @return La reja que hay en la posición @p num
     */
    Reja getReja(int num) const;
};
```

3. Conclusión

Con estas construcciones de estos TDAs seremos capaces de resolver el Sudoku Killer.