Instituto Tecnológico Autónomo de México
Departamento Académico de Computación

“Sudoku”

Proyecto

Larissa Trasviña Ojeda 166280
Emilio Mena Garcia 176115
Alumnos

Estructuras de datos
Materia

Dr. Andrés Gómez de Silva Garza
Profesor

Ciudad de México, martes 24 de abril de 2018

Índice

Descripción del problema	3
Descripción de la solución diseñada	3
Limitaciones de la solución	5
Posibles mejoras y conclusiones	5
Bibliografía	5
Anexo	6

Descripción del problema

Objetivo**:**
Diseñar un sudoku que llene de números del 1 al 9 una cuadrícula de 9x9 celdas dividida en subcuadrículas de 3x3, partiendo de números ingresados por el usuario.

Requisitos**:**
La solución se considera correcta si cumple con las siguientes características:

- Las casillas que el usuario deje en blanco deben regresar un solo número del 1 al 9.
- En una misma subcuadrícula (3x3) no debe haber números repetidos
- en una misma columna y renglón no puede haber números repetido.

Restricciones:
Los límites impuestos a la solución del problema son:

- No deja al usuario usar el teclado para colocar los números en el sudoku, por lo cual no acepta símbolos ni letras que no sean dígitos.
- Si el usuario ya escribió un número, automáticamente el programa no deja colocar el mismo número en la subcuadrícula, fila o columna. Solo si el usuario borra ese número entonces vuelve a estar disponible para colocarlo en otra celda(cuadro).

Descripción de la solución diseñada

Métodos más importantes:

La clase Sudoku tiene todos los atributos, métodos y funciones necesarias para encontrar la solución del sudoku. Utiliza una matriz de enteros para guardar los valores del sudoku. Además, la clase necesita una manera de saber qué valores se han utilizado en cada fila, columna y subcuadrícula. La clase utiliza arreglos de conjuntos (una estructura de datos que guarda valores sin orden y sin repetición) para guardar los valores ingresados por fila, columna y subcuadrícula. Se creó una clase auxiliar llamada *Celda* que guarda dos enteros (una fila y una columna) para facilitar el manejo de esta información dentro del programa.

El método *insertaNumero* toma como parámetros un entero y una celda. Este se encarga de cambiar el valor actual de la celda por el número ingresado y de agregar este número al conjunto de la fila, la columna y la subcuadrícula correspondientes a la celda. Este método es utilizado tanto por la interfaz gráfica (clase encargada de manejar la interacción con el usuario) como por el método *solveSudo* (encargado de encontrar la solución del problema).

El método *solveSudo* se apoya de tres métodos:

- *isFull:* función booleana que regresa verdadero solo si la matriz está llena (no contiene ceros).
- *nextEmptySpace:* regresa en forma de una *Celda* las coordenadas del primer cero en la matriz empezando de la esquina superior izquierda y acabando en la inferior derecha.
- *numerosPosibles:*Toma como parámetro una celda y regresa los números posibles para esa celda. Cada celda pertence a una fila, una columna y una subcuadrícula, entonces los números posibles representan los números del uno al nueve menos los números en la fila, la columna y el subcuadrícula.

SolveSudo utiliza búsqueda exhaustiva recursiva con retroceso apoyándose de los tres métodos descritos arriba. La búsqueda exhaustiva es un método para encontrar la respuesta a un problema dado intentando cada una de las posibilidades hasta llegar a una solución. *SolveSudo* toma como parámetros un entero (número a intentar) y una celda (coordenadas donde se va a insertar el número). A grandes rasgos, el método funciona de la siguiente manera:

- Primero se inserta el número en la celda.
- Después, revisa si ya está llena la matriz (*isFull*).
    - En caso positivo regresa “true”.
    - En caso negativo busca el siguiente espacio vacío (*nextEmptySpace)* y revisa si esta celda tiene números posibles (*numerosPosibles*).
        - En caso negativo, regresa falso y retrocede.
        - En caso positivo intenta sobre esa celda cada uno de los números posibles de manera recursiva.
            - En caso de no encontrar solución después de probar todos los números posibles, retrocede y regresa falso.

<u>Aspectos de diseño:</u>

El sudoku visualmente es una tabla que contiene 81 celdas, divididas en regiones de 3x3 (subcuadrículas). A la izquierda de estas se encuentran los botones de los números del 0 al 9, considerando al 0 como el botón para borrar el número tecleado en esa celda.

A la derecha del 0 se encuentra el botón “*Clear*” que borra todos los números que estén en la cuadrícula ya sean tecleadas por el usuario o el sudoku resuelto.

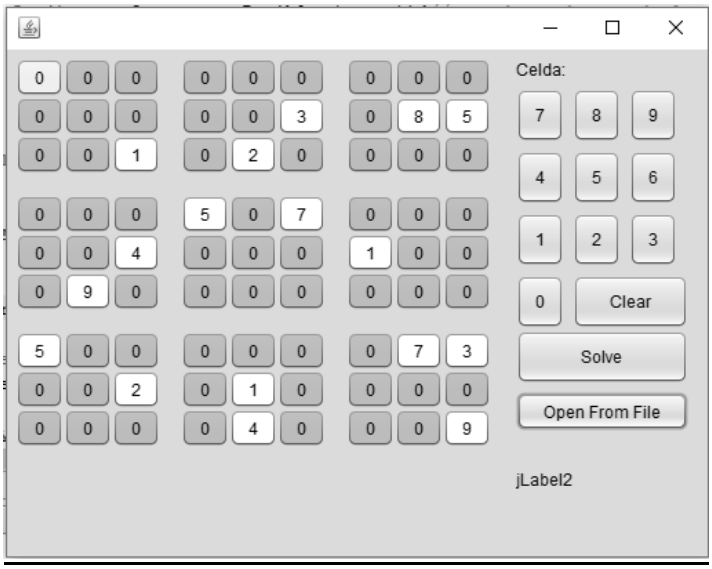
Solve es el botón en el cual al hacer clic resuelve el sudoku tomando los números que el usuario tecleó como base. Abajo de este botón se encuentra el botón “*Open from file*” que permite abrir un sudoku (no lleno) del archivo para resolver.

Las celdas del sudoku están al inicio en 0 y en rojo, al hacer clic en una celda se cambia el color a azul, indicando que es la celda sobre la cual se va a escribir. Al hacer clic en un número (e.g. 9) entonces todos los números iguales a este son de color rojo, para que el usuario vea más fácilmente donde están todos los “9”, análogamente con cualquier otro número del sudoku.

Limitaciones de la solución

Casos no contemplados:

Debido al empleo de búsqueda exhaustiva habrá algunos casos en los que no se encuentre la solución rápidamente, ni tampoco se descarte que pueda tenerse una. Un ejemplo es el siguiente:



Restricciones para usar el programa:

- El programa se debe ejecutar en NetBEeans IDE 8.2 para su correcta funcionalidad.

Posibles mejoras y conclusiones

Una posible mejora al programa sería la implementación de un reloj que indique el tiempo transcurrido desde que se empezó a resolver el sudoku.

Al realizar el proyecto anterior se aprendió a implementar recursividad en una búsqueda exhaustiva. Se solucionaron los errores que se dieron durante las continuas pruebas del programa. Algunos de estos errores tomaron tiempo en solucionarse debido a que se intentó directamente viendo el código, en vez de ejecutar el programa de nuevo y tratar de pensar en una solución más general.

Al ser este un trabajo en equipo se designaron las distintas tareas en función de las habilidades de cada quien. Cuando un integrante no pudo cumplir con dicha parte entonces se externalizó con el otro integrante para poder cumplir con la realización del proyecto.

Bibliografía

Gómez de Silva Garza, A. (s.f.) Consejos para la escritura de métodos recursivos.

(Anexo 1) Diagramas UML: