INSTITUTO TECNOLÓGICO DE COSTA RICA

VICERRECTORÍA DE DOCENCIA
ESCUELA DE INGENIERÍA EN COMPUTACIÓN

CA-3125 ANÁLISIS Y DISEÑO DE ALGORITMOS

DOCUMENTACIÓN CORRESPONDIENTE A LA SOLUCIÓN ENTREGADA PARA EL PRIMER PROYECTO DE CURSO

SOLUCIONADOR DE SUDOKU

PROFESOR:
EDDY RAMÍREZ JIMÉNEZ

AUTORES:
EMMANUEL RODRIGUEZ BEJARANO
KEYLOR D. MUÑOZ SOTO

18 DE OCTUBRE, 2020

I. RESUMEN EJECUTIVO

El objetivo principal de esta tarea consiste en desarrollar una herramienta de software capaz de resolver el juego de rompecabezas, originario del Japón conocido como Sudoku, a través de un algoritmo recursivo que hace uso de pilas y colas para controlar el método de recorrido del árbol de posibles soluciones. En caso de no encontrar una solución al Sudoku evaluado, el algoritmo indica la posición de la celda con menos posibilidades.

Este problema ha sido solucionado a través de una implementación de *backtracking* donde el algoritmo se encarga de generar un árbol de matrices de soluciones y evaluar a cada una de ellas hasta encontrar una que satisfaga los criterios de solución. Para agilizar el proceso de búsqueda, a la matriz principal que contiene el Sudoku se la ha aplicado una máscara de bits de manera que una celda sea capaz de llevar no sólo su estado inicial, sino que también incluye información de sus posibles estados futuros. Esta implementación permite reducir la cantidad de recorridos en la matriz, además de generar y almacenar la información relacionada a la interacción entre celdas durante el paso de inserción, lo cuál, de manera orgánica, genera una matriz Sudoku cuyos elementos sólo deben ser leídos una vez para conocer gran parte de la información necesaria en la resolución del problema.

Una vez concluido el desarrollo del programa, se han llevado a cabo pruebas de tiempo de procesamiento en donde hemos encontrado que el método de solución por recorrido de profundidad encuentra la solución en menos tiempo computacional que el método de recorrido por anchura en la mayoría de los casos que fueron evaluados.

II. INTRODUCCIÓN

El proyecto presentado a continuación consiste en un programa desarrollado en C++ que permite la lectura de matrices de Sudoku desde un archivo de texto. Este programa, además de leer el archivo de texto, debía ser capaz de interpretar el Sudoku presentado y generar una solución, si era posible, para el mismo.

Parte de los requerimientos clave de esta solución era la implementación de pilas y colas para controlar la forma del recorrido del árbol de soluciones, lo cuál permitió establecer un árbol *imaginario* sin necesidad de desarrollar una metodología basada estrictamente en árboles. Es decir, con elementos sencillos, fue posible obtener la funcionalidad que normalmente requería estructuras más complejas gracias a la particularidad del problema.

En este documento se detallan los fundamentos utilizados para la generación de algoritmo. De igual manera, el programa desarrollado y los resultados obtenidos durante la evaluación se detallan en las siguientes secciones.

III. MARCO TEÓRICO

El lenguaje utilizado para realizar este proyecto fue C++ el cual fue diseñado por Bjarne Stroustrup a mediados de los años 80, este fue inspirado por el lenguaje Simula, que es el primer lenguaje relacionado con programación de objetos, vio que era muy útil pero lento y quiso incorporar más funcionalidad con el lenguaje C++ y así terminó mejorando el lenguaje C, el cual es un híbrido por ser un lenguaje orientado a la manipulación de objetos. Su uso empezó desde el año 1983, que fue cuando Rick Mascitti lo utilizo por primera vez.[1]

C++ cuenta con todas las funcionalidades de C y agrega funcionalidades nuevas como clases, sobrecarga de funciones y herencia entre clases. Además, dado que este lenguaje es una extensión de C, a logrado mantener una gran base de usuarios al pasar el tiempo. Entre sus usos se incluyen, pero no se limitan a, la creación de nubes, bibliotecas, bases de datos, herramientas ofimáticas, generación de firmware y sistemas operativos.

Las búsquedas de profundidad y anchura mencionadas en este documento se dan a través de un algoritmo de *backtracking*, cuya traducción al español quiere decir vuelta atrás y es utilizado para identificar de manera recursiva las soluciones que cumplan todas las restricciones necesarias formar parte del conjunto solución del problema.

En términos generales, el *backtracking* se implementa como una función que guarda y vuelve a evaluar posibilidades anteriormente generadas por el mismo proceso de manera que, durante el desarrollo, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa que había sido previamente estimada. Finalmente, cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción.[2]

Existen variaciones de este algoritmo que, a través de la identificación de relaciones con iteraciones anteriores, es capaz de devolverse más de una iteración en el ”árbol”de soluciones, agilizando el proceso de búsqueda de la o las soluciones buscadas.

También, como objetivo principal de este proyecto, se encuentra el uso de pilas y colas como parte de la metodología de resolución. Las pilas y colas son estructuras de almacenamiento de datos cuya principal característica, y a la vez diferencia, es la forma en que los datos son leídos de la estructura misma. En el caso de estructuras de tipo pila, los datos son extraídos bajo el principio *LIFO (last in - first out)* en donde cada siempre se extrae primero el elemento que se insertó de manera más reciente. [3]

Caso contrario, las colas se caracterizan por extraer los datos bajo el principio *FIFO (first in - first out)* en donde los datos se extraen en el mismo orden en que ingresaron, es decir, el primer dato que ingresó es el primer dato en ser extraído.[4]

Este comportamiento es el que permite recorrer el árbol de soluciones por profundidad o por anchura dependiendo del tipo de estructura de almacenamiento que se elige. Por ejemplo, cuando una cola es utilizada, el algoritmo recursivo va a desarrollar las hojas de cada nodo antes que los nodos que se encuentran en el mismo nivel al nodo actualmente analizado. Para el caso de la pila, esta recorre desarrolla primero los nodos de un mismo nivel y posteriormente desarrolla los hijos de éstos.

Algunas de las funciones de pilas y colas que se han utilizado en este programa son:

- push(x): Inserta el elemento x al final de la pila o la cola.
- front(): Busca el dato más antiguo en la cola y lo devuelve.
- top(): Busca el dato más reciente en la pila y lo devuelve.
- pop(): Elimina de la pila o la cola el dato más reciente o el más antiguo respectivamente.

Otra herramienta que fue implementada en el desarrollo del programa fue un enmascaramiento por bits. La mascara de bits es creada para generar identificadores trabajando a nivel de bits agrupándolo en octetos y demás[5], pero lo mas importante del método es el trabajar con ceros y unos, lo cual agiliza las operaciones de comparación[6] que el algoritmo requiere para identificar las posibles soluciones del Sudoku. Cuando se utilizan máscaras de bits, las operaciones sobre ellos son conocidas como operaciones bitwise. Están se desarrollan sobre números binarios en un nivel que reduce la cantidad de iteraciones del procesador para completar una tarea en particular. Las operaciones de *bitshift*[7] y operaciones lógicas de bits [8] fueron las principales herramientas de esta área implementadas en el programa.

En términos de uso de bibliotecas, para agilizar el proceso de desarrollo se utilizo un incluyente general de bibliotecas. Entre las bibliotecas utilizadas destacan las bibliotecas de *iostream* para la recepción e impresión de datos en consola, *string* para el manejo de variables de tipo de cadena de caracteres de tamaño variable, *vector*que se emplea para arreglos y manipulación de datos unidimensionales(arrays) funciona recorriendo la matriz por medio de lectura de vectores.[9], *stack* para la generación de objetos de almacenaje con métodos de pila y *queue* para implementar también los objetos con métodos de cola.

Además, se generaron varias clases que se organizaron en el archivo *clases.h* en el cual se almacenaron los métodos desarrollados [10] para lograr el funcionamiento del programa planteado (la clase Sudoku y la clase Matriz).

Todo el proyecto se desarrolló con el editor de texto Visual Studio Code, el cual es un editor de texto multiplataforma optimizado para diversos lenguajes de programación y de acceso gratuito creado por Microsoft en abril del 2015.[11] Se eligió este editor en particular debido a su facilidad de instalación, subrayado de código y particularidades como una terminal incluida dentro de la misma ventana del editor.

Igualmente, para facilitar el desarrollo remoto, se utilizó la plataforma Github[12], que, a través del software libre de control de cambios Git [13], ayuda a mantener un registro de los cambios desarrollados en los archivos del proyecto y facilitando el compartir el desarrollo y cambios a través de simples comandos de terminal como *pull*, *commit* y *push*.

IV. DESCRIPCIÓN DEL ALGORITMO

En este apartado se demostrará como utilizamos y empleamos los métodos para recorrer el Sudoku por anchura y profundidad por medio de las pilas y las colas, lo cual facilitó la realización del programa.[14]

En la figura 1 se detalla de manera resumida el proceso bajo el cuál funciona el programa desarrollado. Primeramente, el programa lee la entrada de datos que corresponden a la cantidad de Sudoku a solucionar junto con la descripción de cada problema (método de resolución y matriz de números). Durante este proceso la información se guarda en un objeto de clase matriz y a la vez se evalúa si la matriz ingresada es una matriz válida. Es decir, hay más de 16 pistas y ninguna pista tiene conflictos con otra. Además, la matriz debe ser de 9x9.

Seguidamente, el programa genera un primer caso que es la semilla de la función recursiva. Para ello, se encuentra la casilla con menor cantidad de posibilidades y se insertan en la pila o cola n soluciones, es decir, una matriz con una casilla cuyo valor se ha aceptado como uno de los valores posibilidad que tenía esa casilla.

La función recursiva se encarga de obtener un caso de la pila o cola y revisar si este caso es el caso solución o si es posible generar un caso más a partir del caso en análisis. La recursividad continúa hasta que no se pueden crear soluciones, con lo cual se evalúa y se encuentra si se ha llegado al final de una rama con o sin solución.

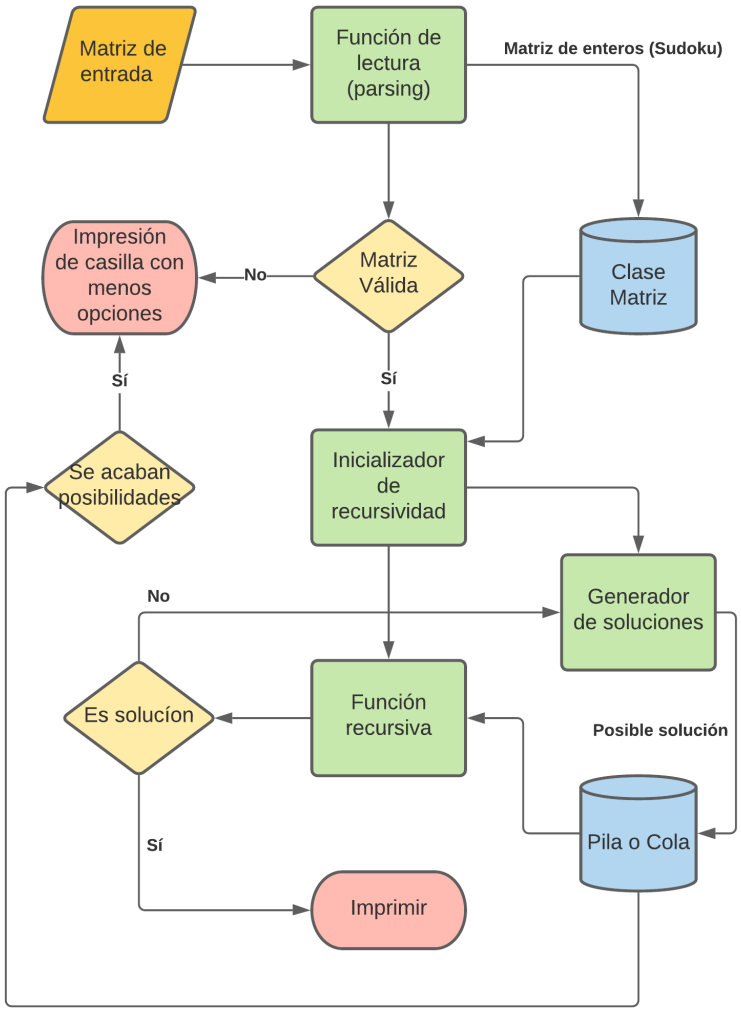

Figura 1: Diagrama de flujo del programa desarrollado.

Durante el proceso de desarrollo encontramos problemas para implementar el sistema de bitmask, pilas y colas. También, en el proceso para leer el archivo de entrada, tuvimos problemas que ocasionaron el cambio del método cin a getline, con la intención de permitir el ingreso de matrices de tamaño dinámico, pero este objetivo no pudo ser alcanzado.

Con respecto a las bitmask, se encontraron problemas al contar y mover los bits de un entero ya que el método tomaba en cuenta que el cero y este interfería a pesar de no ser necesario leerlo. Es por esto que se agrego una linea condicional que excluye al cero y utiliza el bitmask únicamente para los demás números del 1 al 9. Por otra parte, se pensó en guardar mas información codificada en bits superiores, pero esto fue descartado por cuestiones de practicidad y tiempo de

implementación.

Otro problema encontrado durante el desarrollo era el desconocimiento de la manera adecuada de implementar los arreglos de colas y pilas y su relación con los métodos de resolución por profundidad y anchura, fue hasta muy avanzado en el proyecto que se encontró información en las clases impartidas por el profesor e investigación de documentación acerca de pilas, colas, backtracking y su relación entre ellas, permitiendo así su implementación adecuada en el código.

V. RESULTADO DE PRUEBAS

Utilizando la versión final del programa se llevaron a cabo pruebas de tiempo para evaluar e identificar si existían diferencias de efectividad entre los métodos de resolución disponibles. Para esto se generaron 9 archivos de pruebas que contenían distintos Sudoku de diversa dificultad. Además, se incluyó un caso particular que se encontró como "el Sudoku más difícil del mundo"[15] y el Sudoku que originalmente se detalla en el requerimiento de este proyecto.

Para los casos normales, se usaron tres series de grupos de tres Sudoku donde cada uno corresponde a un Sudoku "fácil", "medio" y "difícil". Estos se identifican por la notación de dificultad creciente: **0.txt*, **1.txt*, **2.txt* observable en los nombres de archivo. Cada serie se identifica como *0*.txt*, *1*.txt* y *2*.txt*.

El Sudoku "más difícil del mundo" se detalla en el archivo *33.txt* y el aportado en el requerimiento de este proyecto en el archivo *requerimiento.txt*.[1]

Una vez finalizada la recolección de datos, cuyo resultado se resume en la figura 2, fue posible identificar que el método de resolución por profundidad suele ser más eficiente y requiere menos iteraciones que el método de resolución por anchura. Esto podría deberse al hecho que el método por anchura es capaz de encontrar soluciones más rápido cuando éstas se encuentras en los primeros niveles del árbol de soluciones, pero, contrariamente, le toma muchas más iteraciones que al método de profundidad cuando la solución se encuentra en niveles profundos del árbol tal y como se puede observar en los diagramas desarrollados en Techie Delight [16]. Caso excepción son aquellos Sudoku cuya solución se encuentra en el final de la última rama del árbol de soluciones, ya que para esta particularidad, ambos métodos de resolución necesitarían recorrer el árbol por completo antes de encontrar una solución.

[1] Cada uno de los archivos que contiene estas pruebas ha sido adjuntado en la carpeta inputs contenida en el entregable de este proyecto.

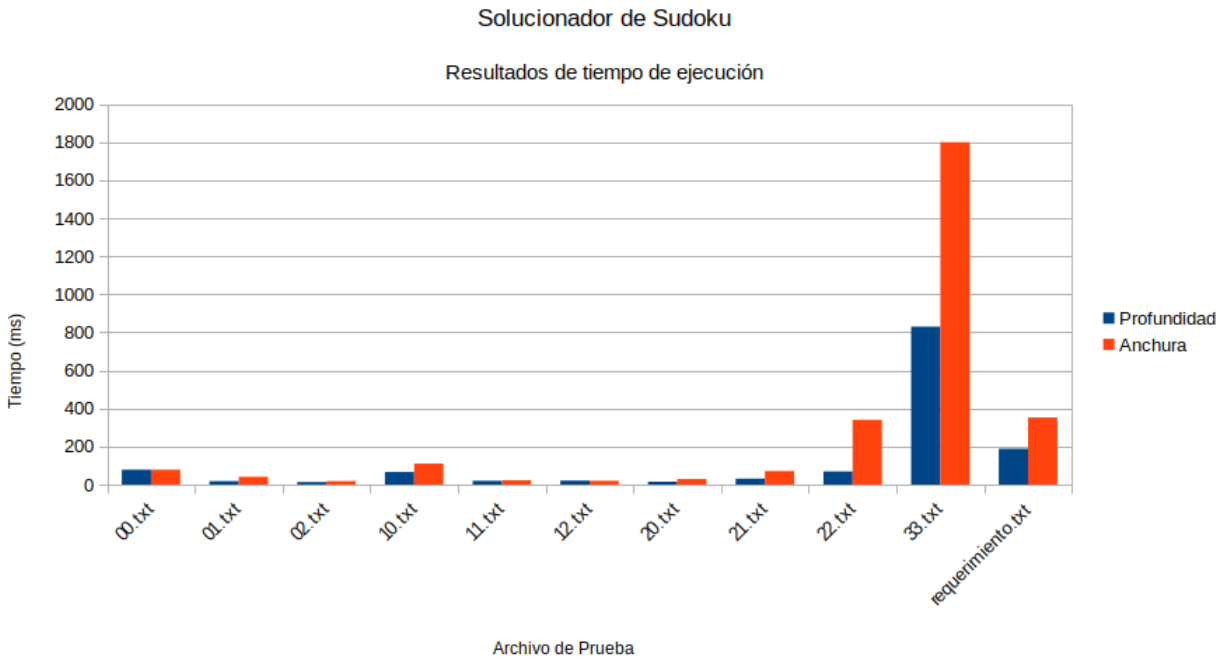

Figura 2: Resultados de tiempo de ejecución del programa para diversos casos de prueba.

Es interesante destacar que “el Sudoku más difícil del mundo” fue el caso que requirió más tiempo de procesamiento, superando hasta en 10 veces algunos de los casos más rápidos encontrados en otras pruebas. Infiriendo a partir de la suposición anterior, podría indicarse que la solución a este Sudoku se encontraba en un nodo profundo, explicando la gran diferencia de tiempo entre el método por anchura y el de profundidad, aunque está hipótesis debe ser primeramente comprobada y su comprobación se sale del ámbito de este proyecto.

VI. CONCLUSIONES

Fue posible desarrollar un programa capaz de resolver una matriz de Sudoku a partir de la implementación de algoritmos de comparación, enmascaramiento de bits y almacenamiento por pilas o colas.

Se comprendieron múltiples términos, conceptos y métodos durante la implementación del código, dado a la vital importancia para su solución, como lo fueron: la mascara de bits, pilas, colas, backtracking, operaciones bitwise y la construcción de bibliotecas personalizadas.

A partir de los resultados obtenidos durante el análisis de rendimiento del programa desarrollado, se encontró que el método de resolución por profundidad es estadísticamente más eficiente que el método de resolución por anchura. Cabe destacar que la solución al problema siempre se encuentra en el último nivel del árbol y, por tanto, el método de anchura sólo puede ser un poco más eficiente que el método de profundidad en contados casos, por lo que se concluye que el método de profundidad es la mejor opción al tratar de resolver esta clase de problemas.

VII. APRENDIZAJES

Keylor

En mi caso este proyecto fue de gran provecho para aprender, en un inicio conocí y desarrolle el uso de la plataformas Github[17] para la transferencia de archivos de una manera mas efectiva con el compañero del proyecto, también el uso de clases y bibliotecas propias, más las utilizadas que son propiamente de C++, realmente fue amplia la investigación para lograr dar con un algoritmo funcional y generar la solución correcta, además de enriquecer mi vocabulario con respecto a la sintaxis de C++ y el como solucionar un Sudoku el cual nunca había resuelto con anterioridad, para lo cual fue necesario la implementación de los métodos de profundidad y anchura, por medio de colas y pilas contando el backtracking.
Todo lo anterior se logro gracias a la búsqueda y de la implementación de la teoría de recursividad y arboles vista en clases.

Emmanuel

Particularmente he encontrado de gran interés la relación que tienen los algoritmos de resolución del Sudoku con los algoritmos de resolución de problemas de restricción y la utilidad que tienen las metodologías de backtraking en este ámbito. Durante el proceso de investigación de este proyecto he aprendido también sobre el uso de estructuras ya existentes en C++ y además he tenido la oportunidad de aprender más sobre operaciones bitwise, el cuál era un área que solía evitar por mi falta de conocimiento y siempre recurría a la utilización de arreglos de matrices o ciclos de comprobación en su lugar.

Otro aspecto importante que he aprendido durante el desarrollo del proyecto ha sido sobre la importancia de aprender a implementar estructuras sencillas antes que estructuras complejas cuando el proyecto lo amerita, como en este caso, que la búsqueda de soluciones a través del árbol se ha llevado a cabo con una herramienta más sencilla.

VIII. BIBLIOGRAFÍA

[1] A. Robledano. Qué es c++. [Online]. Recuperado de: https://openwebinars.net/blog/que-es-cpp/

[2] E. Ramírez-Jiménez. Backtracking. [Online]. Recuperado de: https://www.youtube.com/watch?v=5zBQucjeZ0g

[3] V. Adamchik. Stacks and queues. [Online]. Recuperado de: https://www.cs.cmu.edu/~adamchik/15-121/lectures/StacksandQueues/StacksandQueues.html

[4] A. K. Sharma, *Data Structure Using C*, 1st ed. Pearson India, 2010.

[5] V. Iglesias. Qué son y para qué sirven las máscaras de bits. [Online]. Recuperado de: https://www.victoriglesias.net/mascaras-de-bits/

[6] E. Ramírez-Jiménez. Máscaras de bits ¡con código funcional en c++. [Online]. Recuperado de: https://youtu.be/XgKx3ZV5EoM

[7] IBM. Bitwise left and right shift operators. [Online]. Recuperado de: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzarg/bitshe.htm

[8] Cppreference.com. Arithmetic operators. [Online]. Recuperado de: https://en.cppreference.com/w/cpp/language/operator_arithmetic

[9] O. Palacios. Programación en c++/biblioteca estándar de plantillas/vectores - wikilibros. [Online]. Recuperado de: https://es.wikibooks.org/wiki/Programacin_en_C2B2B/Biblioteca_Estndar_de_Plantillas/Vectores.

[10] Dis.unal.edu.co. Programacion en c++ - clases en c++. [Online]. Recuperado de: https://dis.unal.edu.co/~fgonza/courses/2003/poo/c++.htm

[11] Microsoft. Visual studio code. [Online]. Recuperado de: https://code.visualstudio.com

[12] ——. Github. [Online]. Recuperado de: https://github.com/

[13] L. Torvalds. Git. [Online]. Recuperado de: https://git-scm.com/

[14] W. C. Chavez. Implementación del juego sudoku en c++. [Online]. Recuperado de: https://www.youtube.com/watch?v=br-k9X0bk3k

[15] L. Vanguardia. El sudoku más difícil del mundo, resuelto. [Online]. Recuperado de: https://www.lavanguardia.com/vida/20160527/402090337189/solucion-enigma-sudoku.html

[16] T. Delight. Depth first search (dfs) vs breadth first search (bfs). [Online]. Recuperado de: https://www.techiedelight.com/depth-first-search-dfs-vs-breadth-first-search-bfs/

[17] Github. Hola mundo. [Online]. Recuperado de: https://guides.github.com/activities/hello-world/