Proyecto 1

IC3002 - Análisis de Algoritmos Profe.: Yuen Law Wan I Semestre 2020

Joseph Tenorio Pereira
2019064588

Jose Pablo Muñoz Montero
2019061904

***Abstract*—The following documentation revolves around the creation and analysis of a backtracking algorithm capable of determining the solvability of a given nonogram, and if possible, one of its solutions. The implementation of the created algorithm inside a Unity 2D project is also discussed. Aditionally, the aforementioned Unity project is reviewed as well, and it focuses on the displaying of the given nonogram and its subsequent algorithmic analysis.**

I. Introducción

El *Backtracking* es una técnica de recursión empleada para resolver problemas por etapas, que utiliza como árbol de decisiones a las propias llamadas recursivas que lo conforman. Cuando se "avanza" de etapa se realiza una llamada recursiva, y cuando se "retrocede" se termina el correspondiente proceso recursivo y se vuelve al estado anterior por medio de la pila de llamadas . Esta técnica corresponde a un algoritmo de fuerza bruta, ya que prueba todas las posibles soluciones hasta encontrar la respuesta del problema dado [1]. No obstante, la característica principal de esta técnica es la capacidad de desechar múltiples soluciones simultáneamente cuando se detecta una combinación errónea.

Por otro lado, un nonograma es un tipo de puzzle lógico en el cual se deben rellenar o dejar en blanco los cuadros que conforman una cadrícula dada, la cual posee conjuntos de números en cada una de sus columnas y filas, llamados pistas [2]. Cada uno de los números en las pistas representa la cardinalidad de un grupo de cuadros a rellenar, los cuales obiglatoriamente deben aparecer en su respectiva fila o columna, en el orden indicado con las pistas y separados por uno o más cuadros sin rellenar. El objetivo del juego es rellenar la cuadrícula de modo que se cumplan las restricciones impuestas por todas las pistas, lo cual suele dar como resultado alguna figura de interés.

La forma intuitiva de solucionar un nonograma suele consistir en el rellenado progresivo de cada cuadro y la revisión de si este incumple alguna de las pistas dadas, en cuyo caso de se le descarta como cuadro marcado. Lo anterior se suele complementar con el uso de la lógica matematica para identificar aquellos cuadros que a simple vista deben estar coloreados o en blanco. Tomando en cuenta la anterior definición de *Backtracking*, es claro que la forma intuitiva de soucionar un nonograma se asemeja a una implementación de dicha técnica, y es de esto último de lo que trata el presente proyecto; la creación de un algoritmo de *Backtracking* capaz de calcular una posible solución a un nonograma dado, o bien indicar que dicho nonograma no posee soluciones.

No obstante, el trabajo realizado no se restringe al algoritmo solucionador de nonogramas, ya que se busca construir una aplicación de usuario que implemente dicho algoritmo. La aplicación en cuestión se desarrolla como un proyecto de *Unity 2D* cuyas funcionalidades incluyen la visualización del nonograma procesado, la animación de una cuadrícula que refleje en tiempo real el proceso de *Backtracking*, la entrada de nonogramas por medio de archivos de texto con el formato especificado, entre otras. En adición a lo anterior, se busca que el programa creado funja como herramienta para analizar la eficiencia del algoritmo de *Backtracking*, por lo que el mismo debe mostrar el tiempo de ejecución junto a cada nonograma procesado. Dicho esto, a continuación se documenta la creación de la aplicación propuesta y el algoritmo planteado, así como el respectivo anáslisis de los resultados obtenidos.

II. Trabajo Relacionado

Los puzzles lógicos, tales como los nonogramas o el Sudoku, ya han sido ampliamente estudiados por la ciencia computacional, principalmente con el objetivo de desarrollar algoritmos para su resolución. El Sudoku en particular puede ser resuelto por medio de algoritmos que aplican la técnica de *Backtracking*, como se puede observar en [4]. Dicha implementación del *Backtracking* comparte muchos aspectos conceptuales con aquella correspondiente a los nonogramas. Como evidencia de lo anterior se tiene el hecho de que el algoritmo encontrable en [4] fue utilizado como base del algoritmo desarrollado en este proyecto, el cual comparte estructura general de *Backtracking* con su contraparte para Sudokus.

En cuanto a los nonogramas en sí, estos tambíen han sido el énfasis de muchos proyectos e investigaciones, los cuales suelen tener por objetivo el desarrollo de algoritmos que los solucionen. Los algoritmos resultantes de dichos trabajos no se restringen al *Backtracking*, ya que el mismo se suele combinar con otras técnicas, por ejemplo, los algoritmos genéticos. Sin embargo, proyectos como [3] consiguen crear algoritmos altamente eficientes por medio de la combinación del *Backtracking* con otras herramientas algorítmicas. Dicha combinación consiste en la resolución de la mayor cantidad de cuadros posible antes de ejecutar el *Backtracking*, de modo que este último tenga un árbol de decisión más fácil de recorrer.

No obstante lo anterior, trabajos como [5] muestran algoritmos capaces de solucionar nonogramas con únicamente *Backtracking*, los cuales no suelen ser muy efecientes en comparación con los observables en [3]. Cuando se emplea únicamente *Backtracking* para solucionar nonogramas, las posibles optimizaciones del algoritmo se ven algo limitadas, lo cual es visible al contrastar los trabajos de [5] y [3]. Sin embargo, aspectos como la eficiencia del código que permite verificar la validez de un cuadro (marcado o dejado en blanco) suelen ser los más abiertos al mejoramiento.

III. Métodos

Como ya se mencionó, la solución implementada en este proyecto sigue la técnica de fuerza bruta conocida como *Backtracking* para determinar si un nonograma dado posee o no soluciones, y de poseerlas, determinar una de las posibles respuestas correctas. En adición a lo anterior, el solucionador de Sudokus encontrable en [4] fue empleado como base de la estructura recursiva del algoritmo creado. Además, los nonogramas se representan por medio de dos listas de listas y una matriz numérica. La matriz representa la cuadrícula del puzzle, con cada posición de la misma simbolizando un cuadro. Además, un cero en una cierta posición de la matriz indica que el cuadro respectivo no ha sido determinado; un uno, que el cuadro ha de ser dejado en blanco y un dos que el cuadro ha de ser marcado. Una vez montada la estructura recursiva capaz de operar sobre la representación antes mencionada, solo quedaban dos aspectos del algoritmo por definir: La forma de recorrer los cuadros sin determinar dentro de la matriz y un algoritmo interno capaz de determinar si el estado de un cuadro incumple las restricciones impuestas por las pistas. Cabe resaltar que los mecanismos empleados para realizar las dos tareas antes mencionadas constituyen dos de los principales factores en la eficiencia del algoritmo final.

En cuanto a la verificación de la validez del estado de un cuadro en la matriz, esta es verificada por medio de una función que, en escencia, recorre la columna en la que se ubica el último cuadro alterado (sea para determinarlo como marcado o como en blanco) y la transforma en una lista con el mismo método de codificación empleado en las pistas. Lo anterior quiere decir que cada número en la lista creada representa la cardinalidad de los grupos de números dos encontrados (cuadros marcados) separados por uno más números uno (cuadros dejados en blanco) o números cero (cuadros sin determinar), además, el orden dentro de la lista corresponde al orden de aparación de los grupos. Dicha lista es rellenada con ceros en caso de poseer menos elementos que la pista correspondiente, por el contrario, si posee más elementos, el estado actual de la matriz es determinado como inválido. Una vez creada la lista que representa cada columna, es posible determinar si esta cumple aquellas condiciones que definen si el estado actual de la columna es válido o no. Dichas condiciones dependen de si el último movimiento fue la colocación de un dos o un uno y de las pistas correspondientes, pero en escencia se aseguran de que la cantidad de grupos sea inferior a la cantidad de números en la pista y que la cardinalidad de cada grupo no exceda a su número correspondiente en la pista. Finalmente, se aplica un algoritmo igual a la fila del último movimiento, y si tanto la fila como la columna cumplen las condiciones correspondientes, el estado actual de la matriz es determinado como válido y se procede al siguiente cuadro. En el caso contrario, el estado actual de la matriz es inválido y por lo tanto se revierte a un paso anterior del *Backtracking* por medio de la pila de llamadas recursivas.

Por otro lado, hay dos posibilidades en cuanto a la forma de recorrer los cuadros sin analizar de la matriz, fila por fila o columna por columna. Si bien al principio se teorizó que la influencia del método escogido sobre el tiempo de ejecución era despreciable, luego de múltiples pruebas se determinó lo contrario. La verdad resultó ser que, dependiendo del nonograma en cuestión, el tiempo tomado por el algoritmo para llegar a una solución podía llegar a variar varios milisegundos dependiendo de si se hacía el recorrido fila por fila o columna por columna. En vista de lo anterior, se recurrió al uso de hilos para determinar que recorrido es más eficiente durante el tiempo de ejecución; una vez iniciado el procesamiento del nonograma, se ejecutan dos *Backtrackings* en hilos diferentes, siendo la única diferencia el hecho de que uno de ellos recorre la cuadrícula columna por columna, y el otro fila por fila. El tiempo de ejecución registrado corresponde a la duración del hilo que primero finalice el procesamiento del nonograma. Cabe resaltar que para varios de los nonogramas utilizados durante las pruebas, el tiempo de ejecución con un método u otro variaba considerablemente, a su vez, el tiempo de ejecución menor en ocasiones correspondía al recorrido por filas, y en otras, al recorrdio por columnas

Finalmente, se implementó el algoritmo creado en un proyecto de *Unity2D* que maneja la interfaz gráfica y la carga de los nonogramas a solucionar. Además, dicho proyecto permite vizualizar el tiempo de ejecución registrado y la cuadrícula resultante. Como ya se había mencionado, se añadió la opción de mostrar una animación de la cuadrícula mientras esta es procesada por el algoritmo. Esta última funcionalidad requirió también del uso de hilos, ya que era necesario ejecutar tanto el *Backtracking* como la animación al mismo tiempo, pero separademente.

IV. Análisis de resultados

En cuanto a los resultados obtenidos, cabe resaltar que las pruebas realizadas se enfocaron en nonogramas cuyas dimensiones se ubican entre 10x10 y 30x30. En particular, los nonogramas apreciables en Fig. 1 y 2 fueron los más utilizados durante las pruebas de eficiencia, y es por medio de los cuales se midió el progreso en la optimización del algoritmo. Como se puede observar, fue posible reducir el tiempo de ejecución del nonograma en Fig. 1 a un rango de entre 80 y 60 milisegundos, aunque cabe resaltar que dicho rango puede variar dependiendo del equipo uitlizado para ejecutar el proyecto. También cabe mencionar que el primer algoritmo funcional que se tuvo tardaba aproximadamente 18 segundos en resolver este mismo nonograma, por lo que se

puede decir que hubo una mejora de aproximadamente 17 000 milisegundos para este caso.

Fig. 1. Tiempo de ejecución y cuadrícula del principal nonograma de prueba en *Unity*.

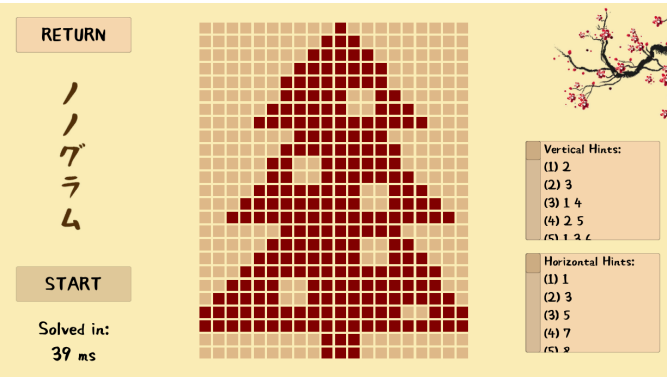

Fig. 2. Tiempo de ejecución y cuadrícula de un nonograma de prueba en *Unity*.

En cuanto al nonograma en Fig. 2, la resolución del mismo toma entre 45 y 25 milisegundos. Como dato adicional, este nonograma fue una de las principales razones por las que se implementaron las distintas formas de recorrer los cuadros sin analizar de la matriz. Lo anterior se debe a que si se realizaba únicamente el recorrido por columnas, el tiempo de ejecución con este nonograma aumentaba significativamente, hasta a 1000 milisegundos, mientras que con el recorrido por filas disminuía a los valores obtenidos en la versión final.

En Fig. 3 y 4 se pueden apreciar las cuadrículas y tiempos de ejecución de las soluciones a otros nonogramas. Cabe mencionar que el algoritmo también fue expuesto a nonogramas sin respuesta correcta, y en todas las pruebas realizadas el algoritmo indicó correctamente la carencia de una solución, asimismo, siempre se dió una posible solución a aquellos nonogramas que la tuvieran. El nonograma de prueba en Fig. 4 tuvo el tiempo de ejecución más alto de todos los nonogramas utilizados.

V. Conclusión

En conclusión, queda probado que el algoritmo desarrollado es siempre capaz de determinar si el nonograma dado

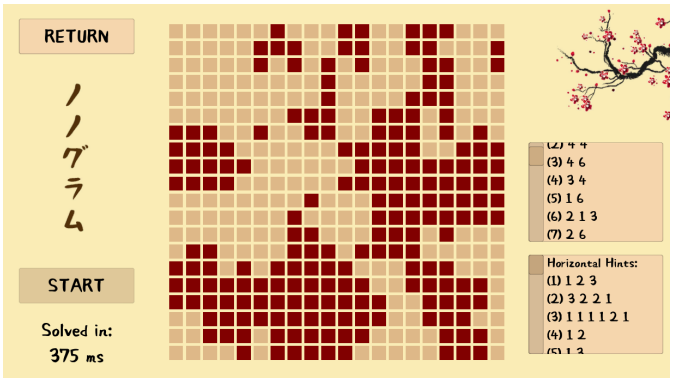

Fig. 3. Tiempo de ejecución y cuadrícula de un nonograma de prueba en *Unity*.

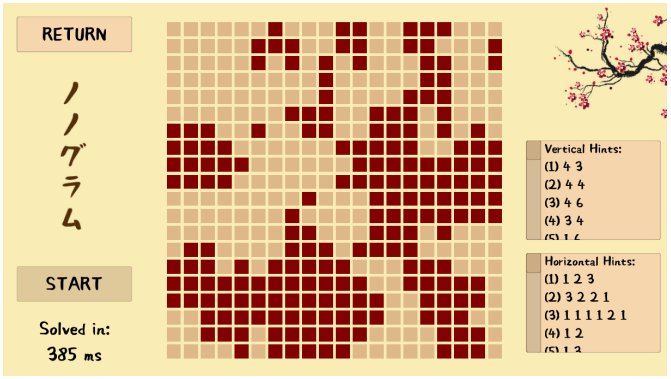

Fig. 4. Tiempo de ejecución y cuadrícula textitUnity.

posee o no solución. Lo anterior aplica, como minímo, a nonogramas cuyas dimensiones están entre 10x10 y 30x30, si bien no hay nada que sugiera que el algoritmo vaya a tener dificultades con nonogramas de mayor tamaño. En cuanto al tiempo de ejecución, durante las pruebas se pudo observar que aún para nonogramas con dimensiones similares, factores como el equipo utilizado y las pistas dadas hacen que el tiempo resultante varíe ampliamente. No obstante, un aspecto a destacar fue que entre las primeras versiones del código y la versión final del mismo hubo muhísima optimización del tiempo de ejecución.

Por otro lado, se toma como un éxito al proyecto de *Unity* en el que se implementó la solución creada, ya que el mismo ofrece una interfaz amigable con el usuario. Además, el proyecto creado facilita la carga de nonogramas a partir de archivos de texto con el formato especificado y la muestra de los aspectos relevantes del procesado del nonograma: las pistas dadas, la cuadrícula resultante y el tiempo de ejecución. Finalmente, la opción de animar la cuadrícula durante la corrida del algoritmo también se añadió satisfactoriamente.

VI. Acceso al proyecto

El repositorio de GitHub en el cual se ubica el proyecto de *Unity* creado puede ser accesado mediante el siguiente link: https://github.com/TilapiaBoi/AnalisisPRY1

BIBLIOGRAFÍA

[1] J. Lázaro, *Backtracking*, Departamento de Ciencias de la Computación, Universidad de Alcalá, n.d. Accessed on: Mar. 18, 2020. [Online]. Available: ftp://www.cc.uah.es/pub/Alumnos/G_Ing_Informatica/Algoritmia_y_Complejidad/anteriores/Apuntes/08_Backtracking.pdf

[2] "Nonogram", n.d. Accessed on: Mar. 18, 2020. [Online]. Available: https://www.definitions.net/definition/Nonogram

[3] C. Wu , D. Sun, L. Chen, K. Chen, C. Kuo, H. Kang and H. Lin, "An Efficient Approach to Solving Nonograms", *IEEE Transactions On Computational Intelligence And AI In Games*, vol. 5. no. 3, Sept. 2013, pp. 251-264. Accessed on: Mar. 18, 2020. [Online]. Available: http://perpustakaan.unitomo.ac.id/repository/AnEfficientApproachtoSolvingNonograms.pdf

[4] Tech With Tim, *Python Sudoku Solver Tutorial with Backtracking p.2*, Apr. 4, 2019. Accessed on: Mar. 18, 2020. [Video file]. Available: https://www.youtube.com/watch?v=lK4N8E6uNr4&t=844s

[5] M. Richards, *Backtracking Algorithms in MCPL using Bit Patterns and Recursion*, Computer Laboratory, University of Cambridge, Feb. 23, 2009. Accessed on: Mar. 18, 2020. [Online]. Available: https://www.cl.cam.ac.uk/~mr10/backtrk.pdf