**INFORME DEL**

**PROYECTO SUDOKU HIDATO**

**Asignatura:** Programación Declarativa

**Integrantes:**

David Manuel García Aguilera

Rolando Sánchez Ramos

**Grupo:** C311

INTRODUCCIÓN:

**Descripción del Juego:**  
En el Sudoku Hidato, objetivo es rellenar el tablero con números consecutivos que se conectan horizontal, vertical o diagonalmente. En cada juego de Hidato, los números mayor y menor están marcados en el tablero. Todos los números consecutivos están adyacentes de forma vertical, horizontal o diagonal. Hay algunos números más en el tablero para ayudar a dirigir al jugador sobre cómo empezar a resolverlo y para asegurarse de que ese Hidato tiene solución única. Se suele jugar en una cuadrícula como Sudoku pero también existen tableros  
hexagonales u otros más irregulares con figuras como corazones, calaveras, etc. Cada puzzle de Hidato creado correctamente debe tener solución única.

**Propuesta del proyecto:**

Para la realización de este proyecto se requerirá la implementación de dos programas o módulos principales utilizando el lenguaje de programación Haskell. El primero tendrá como objetivo generar Hidatos válidos con solución única y atendiendo a las restricciones planteadas anteriormente. El segundo se encargará de solucionar un Hidato dado y analizar si tiene o no solución o si tiene al menos más de una.

DESARROLLO:

**Definición de los Hidatos y herramientas auxiliares:**  
Para la definición de un *Hidato* se definió un record con este nombre el cual presenta dos constructores, uno denominado *NilHidato* (referente a *Hidatos* nulos) y otro denominado *Hidato* que contiene como campos:

* *board*: Una matriz que contiene los valores -1 (para celdas inválidas), 0 (para celdas válidas sin un número colocado) y números entre 1 y N (cantidad de casillas válidas del tablero).
* *posMin* y *posMax*: Son campos del tipo definido *Pos* y que representan la posición del mínimo y el máximo en las casillas válidas con números colocados en el tablero correspondiente.
* *minValue* y *maxValue*: Campos de tipo *Int* que contienen los respectivos valores máximo y mínimo de la matriz del *Hidato* dado que están en las posiciones válidas con un número colocado

También es importante mencionar que para trabajar con más facilidad las posiciones y las referencias a estas en los hidatos y métodos auxiliares se implementó el tipo record *Pos* el cual contiene dos campos, *row* y *column*, los cuales hacen referencia a un par (fila, columna) de un *Hidato* determinado. Además, este tipo tiene un constructor *NilPos* utilizado para definir una posición nula.

Para trabajar sobre el campo *board* del tipo *Hidato* se crearon las funciones update\_matrix para hacer cambios sobre este y el método *printMatrix* para mostrar al mismo.

**Solución de un Hidato:**  
Para lograr solucionar un elemento de tipo *Hidato* se implementó el módulo *Solver.hs* el cual contiene a las funciones:

* *solve*: Recibe un tablero en forma de matriz y devuelve una lista de tableros solucionados el cual esta diseñado para tener 0, 1 o 2 soluciones. Con este método se puede validar si el tablero dado tiene solución única o detectar de manera temprana cualquier caso contrario. El método en esencia realiza un *backtracking* con varias podas para verificar las posibilidades de resolución con las restricciones dadas.
* *backtrack*: El método principal que se encarga de dado un estado de un tablero y una posición dada trata de hacer los llamados recursivos correspondientes a los subproblemas que generan las posiciones adyacentes a la actual. Las podas bases son: conexidad del grafo o si la cantidad de soluciones generadas fueron 2 diferentes o ninguna.

**Generación de un Hidato válido con solución única:**  
Para generar un hidato válido de solución única se implementó el módulo *HamiltonianTourGenerator.hs* el cual contiene tres métodos principales:

* *generateSolvedHidato*: Este método recibe una matriz con las condiciones definidas para una matriz de un hidato definido por el usuario, una posición válida para empezar un camino Hamiltoniano desde esta y recorriendo todas las celdas válidas vacías en el tablero y finalmente también recibe las dimensiones de la matriz. Como retorno se da un elemento de tipo *Hidato* en cuyo campo *board* se encuentra una matriz generada con un camino Hamiltoniano marcado por una secuencia desde 1 hasta N (número total de celdas válidas vacías a rellenar). Esta función cuenta además con dos casos, uno cuando no es posible hallar un camino Hamiltoniano en la matriz dada y otra cuando se logra generar este camino en la matriz inicial y por lo tanto se procede a crear un *Hidato*. Para crear un hidato y rellenar sus campos se utilizan los métodos *hamiltonianPath*, *getPosMinBoardGZ*, *getPosMaxBoardGZ, getMaxValueGZ* y *getMinValueGZ*,los cuales hallan el camino Hamiltoniano, la posición del máximo y el mínimo valor mayor o igual que 0 y los propios valores máximo y mínimo respectivamente.
* *hamiltonianPath*: Método que recibe una matriz de entrada con una posición actual para empezar a construir un camino de Hamilton y devuelve la matriz con el camino dado en el orden de los números de 1 a N colocados, y en caso de no ser posible encontrar un camino Hamiltoniano se devuelve una matriz con el valor -1 en cada casilla. La esencia de este algoritmo es un *backtracking* mediante *DFS* y que en cada iteración trata de, para la posición acual en análisis, ver sus casillas adyacentes no analizadas y llamar a este método pasándole el tablero con un -1 en la posición del llamado padre y que recibe como posición a analizar una de las adyacentes seleccionadas. Además, los casos base de este algoritmo y las podas se basan en si el grafo generado por la matriz actual es conexo o si se llegó a una posición donde esta es la única posible a completar en todo el tablero. En otro caso se realiza la idea recursiva explicada anteriormente y dado que el retorno en cada llamado devuelve un camino hamiltoniano posible, si se encuentra un camino se suma 1 a todos los valores de este y en la celda actual se pone un 1.
* *generateHidatoWithUniqueSolution*: Se encarga de, a partir de un hidato resuelto que obtiene como entrada, devolver el mismo hidato pero con posiciones faltantes, pero de tal manera que solo exista una forma de completarlo nuevamente y que cumpla con las reglas de validez de hidatos. La esencia es llamar a otro método auxiliar que, a partir de un hidato y un conjunto de posiciones validas a quitar, toma una de manera aleatoria y analiza el hidato resultante con el método *solve* de *Solver.hs*.¡ y si este tiene una solución única, quita dicha posición del hidato y del conjunto de posiciones válidas a quitar y se llama al mismo método en el nuevo tablero con las nuevas posiciones. Este proceso terminar si al quitar una posición se obtienen 0 o 2 soluciones utilizando el solucionador.

**Estructura del proyecto e interacción entre módulos:**  
El proyecto fue creado a partir de la herramienta *Stack* la cual es ampliamente utilizada en la creación de proyectos de Haskell. Con esta se logra crear un proyecto con el comando *stack new <Nombre del proyecto>* y luego el mismo se compila y se corre con los comandos *build* y *run* respectivamente.

En el proyecto existen dos carpetas fundamentales:

1. *src:* Que contiene los módulos *HamiltonianTourGenerator.hs*, *Solver.hs* y *Utils.hs* los cuales contienen todas las herramientas necesarias para el funcionamiento del proyecto.
2. *app:* Contiene el script *Main.hs*, en el cual se ejecuta el núcleo de la aplicación a partir de los módulos definidos en *src.*

Para el funcionamiento de los módulos para generar y resolver Hidatos se utiliza el módulo *Utils.hs* mediante el cual se satisfacen todas las operaciones y funciones auxiliares requeridas en el proyecto. Por otro lado, el módulo *HamiltonianTourGenerator.hs* importa de *Solver.hs* el método *solve*, el cual como ya se mencionó anteriormente, permite verificar la unicidad y validez de la solución de un hidato determinado.

**Ejecución y uso de la aplicación:**  
Una vez se ejecuten los comandos de compilación y ejecución mencionados, se genera un archivo ejecutable denominado *HaskellHidato-exe.exe*, el cual al ser ejecutado desde un entorno de Windows genera una aplicación de consola con el siguiente flujo:

1. Se le da la bienvenida al usuario y se le pide que entre por consola la opción que desea realizar con la app.
2. Al seleccionar una opción (1 o 2), si se seleccionó la 1 entonces se le pide al usuario que entre por consola unos enteros n y m representando las dimensiones de un hidato rectangular completamente vacío para ser generado y resuelto. Por otro lado, si el usuario selecciona la opción 2 se procede a pedirle que entre las dimensiones de un hidato personalizado por este, por lo que debe entrar manualmente por consola los elementos de la matriz del hidato deseado.
3. Finalmente, se le imprime al usuario la forma del hidato dado sin resolver, el cual tiene algunas posiciones que faltan y luego se imprime el mismo hidato resuelto. Luego de esto finaliza la app y se muestra un mensaje de despedida al usuario.

CONCLUSIONES:

Con la realización de este trabajo de demuestra la facilidad y legibilidad del código de un lenguaje declarativo y funcional como Haskell para resolver problemas difíciles de implementar de forma entendible directamente a partir de una vía imperativa. Además, se evidencia la consistencia de Haskell como lenguaje capaz de realizar complejas operaciones basadas en funciones de orden superior y la recursividad. Finalmente, consideramos ampliamente interesante el empleo de este lenguaje para enfrentarnos a un problema en la categoría NP, como lo es la resolución de hidatos y generación de estos con solución única.