Informe Entrega 1

Alumno: Benjamín Zúñiga

Profesor: Pablo Román

Asignatura: Taller de Programación

Abril 2024

Introducción

En este informe se verá como es posible crear un programa capaz de resolver un “15 puzzle” de manera automática con una cantidad relativamente optima de pasos y en un tiempo considerablemente bajo para el tipo de problema que se está tratando, todo lo anterior gracias a la implementación el algoritmo “A Star”, “A estrella” o “A*”, la creación de estructuras de datos eficientes y la implementación de una heurística adecuada.

Solución

Para la solución del problema mencionado fue necesario primeramente implementar el algoritmo A* el cual consiste básicamente en partir de un estado inicial, el cual en este caso es un tablero que contiene números desordenados desde el 0 hasta el largo de una fila o columna al cuadrado menos 1, e ir generando los posibles movimientos hasta llegar al estado solución.

Implementación

Con respecto a la implementación de la solución, esta fue desarrollada mediante la creación de 5 clases principalmente, las cuales fueron “State”, “Heap”, “StackNode”, “Stack” y “Puzzle”, donde además fue creado un archivo main el cual es el ejecutable que se usa para instanciar el programa. Hablando en más detalle de cada clase, la principal de este programa es “State”, ya que esta representa las distintas configuraciones de tableros que se van creando a medida se va buscando la solución, además, esta clase incluye atributos como el número de la fila y columna donde se encuentra el 0, un puntero al estado “padre” del actual, ids que actúan de identificadores para no comparar posición a posición cada tablero, los pasos que se han hecho para llegar al estado, la cantidad de números fuera de su posición, la cantidad de pasos faltantes, la cual se calcula mediante “distancia de manhattan” y conflictos lineales, de donde al ponderar por algunos valores escogidos después de algunos ensayos con distintos tableros forman la heurística utilizada para calcular el “peso” de cada estado para darle más o menos prioridad a la hora de ordenarlo en el heap. Luego está la clase Heap, la cual es la implementación de un heap mínimo, ya que esta es una estructura bastante eficiente a la hora de necesitar una cola de prioridad, debido a que siempre que se hace “pop” de un elemento, el elemento extraído es el mínimo en el caso del heap mínimo, y a la hora de insertar un elemento, el tiempo de esta acción es más eficiente que el que se vería en una lista normal.

Con respecto a la clase StackNode y Stack, sirven para formar una lista enlazada, donde StackNode es cada nodo de la lista, donde se guardan solamente los ids de cada tablero, y Stack es un puntero a la cabeza de la lista. Por último, la clase Puzzle es la cual ejecuta el algoritmo estrella (A*) y esta para poder ejecutar el algoritmo estrella, contiene como atributo el estado actual, sobre el cual se va a operar, y dos conjuntos, uno llamado abierto, el cual contiene los estados que falta por revisar y está implementado mediante un Heap, y otro llamado cerrado, el cual contiene los estados ya revisados, para así evitar repeticiones a la hora de agregar estados al conjunto abierto, este conjunto fue implementado con una lista enlazada.

Consideraciones

El proyecto se realizó y comprobó la mayoría del tiempo en el sistema operativo Windows, donde, aunque se probó el programa en Ubuntu 23.10.1 y compiló el código sin problemas, resultaron diferencias en algunos resultados observados, principalmente en el tiempo de solución de los tableros 5x5, donde, por ejemplo, el tablero con el que se evaluará el bonus en Windows se demora entre 12 y 13 segundos en responder al programa, mientras que en Ubuntu se demora 2 minutos.

Conclusión

Finalmente se puede concluir que se logó dar solución al problema planteado que era generar una especie de “Optimal Solver” de puzzle 15 mediante el algoritmo estrella, pero a pesar de esto, aunque los resultados son satisfactorios y relativamente buenos en cuanto a tiempo y la cantidad de pasos, gracias principalmente a la heurística implementada, a la hora de ir agrandando el tamaño de los tableros el programa empieza a demorarse más, posiblemente a la lista enlazada, ya que el número de estados que contiene aumenta de gran manera, haciendo que buscar si un elemento ya fue creado es más costoso.