UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Informe de Tarea 1 – Aplicación del algoritmo A* para la resolución del 15-puzzle

Alumno: Mohamed Al-Marzuk

Curso: Taller de Programación

Profesor: Pablo Román

Explicación del problema

El problema consiste en la resolución de un juego llamado 15-puzzle utilizando el algoritmo A*. Para esto, se utilizan matrices cuadradas, con un tamaño desde n = 1 hasta n = 5, numeradas de forma “aleatoria” desde el 0 hasta n*n-1, siendo el 0 el espacio en blanco, es decir, la única casilla que se puede mover en las cuatro direcciones. El algoritmo utiliza estados para representar la posición actual, manteniendo un registro del tablero, su tamaño, la posición del 0, el estado padre (desde el cual se llegó a la posición actual), la cantidad de movimientos acumulados para llegar a la posición actual, y el cálculo de la heurística, esto último es para determinar el costo de llegar a dicha posición, comparándola con el estado objetivo de los números ordenados de forma correcta. El algoritmo usa una cola de prioridad con un heap min, para asegurarse de que siempre se escoja el estado menos costoso, basado en su heurística. El objetivo final es ordenar el tablero en una cantidad óptima de pasos.

El estado inicial se genera a partir de un archivo con extensión .txt en donde se encuentra la matriz inicial. El algoritmo genera por sí solo la matriz objetivo, dependiendo del tamaño de la matriz leída, y luego entra en un ciclo que va revisando cada uno de los posibles movimientos hasta encontrar una solución, si es que dicha solución existe. Una vez encontrada la solución, se muestra el camino encontrado, junto a la cantidad de pasos demorados y el tiempo de ejecución. Si no encuentra solución, muestra por pantalla que dicha solución no existe. El algoritmo utiliza dos arreglos de Heap, uno para los abiertos, en donde se guardan los movimientos a explorar, y otro para todos los estados ya explorados.

Heurística

Para resolver este problema, se utilizó el algoritmo A*, el cual busca el camino más corto entre dos estados. Para realizar esto, se utiliza una cola de prioridad, enfocándose siempre en los estados con menor valor heurístico.

Como se mencionó anteriormente, la estructura utilizada para la cola de prioridad es un árbol binario de tipo heap min, en donde cada padre es menor que sus dos hijos. De esta forma, podemos asegurar que el estado que esté en el tope de la cola (aquel que se obtiene con pop()) siempre será el de menor heurística, y por lo tanto, el que tenga más posibilidades de llevar a una solución.

La heurística utilizada para ordenar esta cola de prioridad considera la distancia manhattan entre dos estados y el conflicto lineal entre ambos. De esta manera, se consideran los estados con menor suma entre distancia manhattan y conflicto lineal.

Aspectos de implementación

Para la implementación de este problema, se pueden destacar varios factores que ayudan a mejorar la eficiencia de la búsqueda de una solución.

- **Uso de A***: Este algoritmo es conocido por combinar técnicas de búsqueda como la búsqueda en anchura (BFS) y la utilización de heurística para estimar el coste desde el estado actual hacia la meta. Siempre escoge el estado con menor valor heurístico.
- **Distancia Manhattan**: Es una de las dos técnicas heurísticas utilizadas en la resolución de este problema. Se define como la diferencia absoluta entre dos números. Para el caso de dos tableros, se calcula la diferencia absoluta entre cada par de números y se suman todas esas diferencias.
- **Conflicto Lineal**: Es la segunda técnica heurística utilizada. Se define como la búsqueda de cada par de números en una fila o columna que deben intercambiar sus posiciones para alcanzar su posición objetivo. Este valor se suma a la distancia manhattan.
- **Cola de prioridad**: La utilización de un árbol de tipo Heap asegura que se pueda extraer el estado con la menor heurística, acelerando el proceso de búsqueda.

La combinación de todo esto garantiza una implementación eficiente que permita encontrar la solución en la menor cantidad de pasos posibles y en un tiempo reducido, evitando caminos innecesarios.

Diseño del programa

El diseño consiste en la creación de las clases necesarias para cumplir con lo requerido por el problema. Para esto hay que identificar qué aspectos son fundamentales para su implementación, a continuación se brindará detalles sobre esto:

- **Estado**: Es la clase mínima necesaria para construir el resto. Contiene un tablero, un tamaño, la posición actual del espacio en blanco (0), una referencia al padre, un contador de movimientos hasta llegar a la posición actual y un valor heurístico. Dentro de esta clase están las funciones de cálculo de la heurística, las funciones de movimiento, la función de expansión y las de printeo.
- **Heap**: Esta clase es la que se utiliza para la cola de prioridad. Contiene un arreglo de punteros a estado, un valor de capacidad máxima y un valor de cantidad de estados actuales en el arreglo. Esta clase implementa las funciones de push y pop.
- **Puzzle**: Esta clase es la que implementa el algoritmo A* como tal. Contiene dos estados, uno inicial y uno final (con los números ordenados correctamente), además

contiene dos arreglos Heap, uno con los estados abiertos y otro con todos los estados visitados. La función solve es la que opera A*.

Funcionamiento del programa

A continuación se brindará un paso a paso de cómo funciona el programa:

1. El programa lee el archivo indicado por terminal y crea una matriz correspondiente a lo que lee en ese archivo. Luego, crea un estado y lo asigna como inicial dentro del puzzle. Finalmente, genera un estado final dependiendo del tamaño de la matriz
2. Tras esto, se hace un llamado a la función solve, esta función pushea el estado inicial dentro de ambos arreglos Heap, y entra en un bucle que permanecerá activo siempre que haya estados disponibles dentro del Heap de abiertos, e irá comprobando si el estado actual es la solución, si no lo es, lo pushea al Heap de todos los estados y lo expande, es decir, encuentra todos los posibles movimientos a partir de ese estado, y comprueba si esos estados no están en ninguno de los arreglos heap, si se cumple esta condición, calcula su heurística y los pushea a los abiertos.
3. Se extrae el estado más optimo, es decir, el de menor heurística, y ese queda como estado actual, si no es la solución, vuelve a hacer las comprobaciones mencionadas.
4. Si encuentra solución, imprime todo el recorrido realizado hasta llegar a ese estado, junto a la cantidad de pasos y el tiempo de ejecución.
5. Si no encuentra solución, imprime por pantalla que no ha sido encontrada.

A continuación se presenta una tabla con cada uno de los tableros, su tiempo de ejecución y los movimientos realizados.

Tablero ejecutado	**Movimientos realizados**	**Tiempo (segundos)**
Tablero1.txt	8	0,001029
Tablero2.txt	15	0,000473
Tablero3.txt	24	0,001457
Tablero4.txt	3	0,00045
Tablero5.txt	47	0,001274
Tablero6.txt	154	0,025023
Tablero7.txt	130	0,015733
Tablero8.txt	212	1,67022

Ejecución del código

Para poder ejecutar el código, es necesario utilizar el sistema operativo Linux, y asegurarse de que todos los archivos .h, .cpp y .txt estén en un mismo código, al igual que el archivo makefile.

1. **Preparación del entorno**: Abra una terminal Linux y navegue hasta el directorio de los archivos.
2. **Compilación:** Ejecute el archivo make para compilar y crear todos los archivos necesarios para la ejecución del programa.
3. **Ejecución:** Invoque el comando ./main, y ahí se desplegará un menú. Pulse 1 para resolver un puzzle, y escriba su nombre por la terminal.
4. **Resultados:** Si el archivo existe, se mostrará el recorrido de la solución, los pasos demorados y el tiempo de ejecución

Bibliografía

Implementing A-star(A) to solve N-Puzzle*. (2016, 3 mayo). Insight Into Programming Algorithms. https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle/ (*Implementing A-star(A*) To Solve N-Puzzle*, 2016)