Taller N°1

Puzzle 15 Generalizado

usach

Vicente Gerardo Arce Palacios

Taller de Programación 2024-1

08 de abril 2024

Explicación breve del algoritmo

Para resolver el problema del puzzle 15 generalizado, se utiliza el algoritmo A*, utilizando una matriz de dimensiones n = 1, 2, 3, 4 y 5. En esta matriz cada posición representa una pieza del tablero, siendo el 0 la representación abstracta del espacio vacío. Abordando el funcionamiento del algoritmo A*, este en palabras simples ayuda a la resolución de cualquier tipo de problema que requiera cierta serie de pasos, realizando todos los pasos posibles para un cierto estado, continuando así con el caso más favorable en relación con una o más heurísticas escogidas, esto último de poder reconocer que paso a seguir es más favorable que los demás hace que este algoritmo encuentre la solución óptima de la forma más rápida posible en la mayoría de los casos.

Los estados, son usados para mantener un registro de como está el tablero en cierta etapa de ejecución del algoritmo, conteniendo estos estados la información del tablero actual (disposición de las piezas), el peso que tiene dicho estado (heurística) y demás atributos que se detallarán en la sección “Funcionamiento del programa”.

Heurística o técnicas utilizadas

Las heurísticas usadas para el algoritmo A* en el caso de esta implementación fueron tres, siendo estas la cantidad de piezas que no estaban en la posición correcta, la distancia Manhattan con conflicto lineal y por último la profundidad a la que cada estado se encuentra (cantidad de pasos hasta llegar a dicho estado). Además, a estas heurísticas se le agregaron ciertos ponderadores, los cuales en el código se representan por pMT, pMDLC y pD. Estos fueron agregados con la finalidad de darle mayor importancia a ciertas heurísticas antes que a otras dependiendo su relevancia del tamaño del tablero, estos ponderadores se establecieron por prueba y error, llegándose a la conclusión que los elegidos son los que permiten llegar a una solución óptima y eficiente en tiempo computacional.

Como estructura de datos para almacenar los estados que se iban generando y para luego ir escogiendo el óptimo, se seleccionó un árbol tipo Heap. Esta estructura de datos se escogió debido a que esta ordena los elementos de tal forma que la heurística siempre se ubique en la primera posición del árbol, es decir que, al hacer un pop, siempre se obtendrá el estado que más chances tiene de llevar conducir a la solución.

Con lo ya mencionado se garantiza que se priorizaran aquellos estados que tengan una menor heurística, es decir, se lograra obtener de manera rápida el estado con menor heurística ya que este se encuentra en el tope del Heap.

Funcionamiento del programa

El problema se representa por un “Estado”, este se compone de dos matrices de enteros, la primera representa el tablero y la segunda el tablero objetivo; a su vez el estado tiene número de filas y de columnas, su peso (heurística), la profundidad del estado (cantidad de pasos que se tuvieron que realizar desde el estado inicial hasta llegar a dicho estado), un par de enteros que representan las coordenadas de la posición vacía del tablero, un puntero al padre del estado y un arreglo de hijos que tiene el estado (doble puntero).

Para gestionar estos estados se empleó un Heap que ordena los estados según la heurística de los nodos, siendo el estado con menor nodo el que se extraerá siempre al ocupar la funcionalidad “pop”.

El programa inicia leyendo una matriz desde un archivo txt dado por el usuario que ejecuta el programa. Leyendo esta matriz se inicializan los atributos del estado y se definen los valores de los ponderadores con relación a la dimensión de la matriz, teniéndose así la base para empezar con el algoritmo A*. Luego de esto se crea un objeto del tipo “State *” el cual permitirá inicializar un objeto del tipo “AStarAlgortihm” en el cual se carga el estado inicial como uno de sus atributos para luego aplicarle el algoritmo “AStar”.

Dentro de la función A* se inicializa el peso del estado inicial y luego se introduce al Heap de los estados abiertos (estados sin visitar), entrándose así a un bucle. Mientras queden estados en el Heap de abiertos el algoritmo saca el primer estado, si no es el estado final, entonces se agrega a la lista de los cerrados el estado y luego realiza todos los posibles movimientos para este estado (hijos), luego se calcula el peso de cada uno de los estados creados y se agregan a la lista de abiertos siempre y cuando estos estados no estén previamente en el Heap de los abiertos y de los cerrados. Este proceso se repite hasta llegar a la solución, devolviéndose así un puntero al estado que contiene el resultado final, finalmente se imprime el camino óptimo para llegar a la solución y el tiempo que se requirió.

Aspectos de implementación y eficiencia

El código es eficiente por los siguientes pilares fundamentales:

- Heurística: las heurísticas usadas calculan de manera eficiente que estados son mayormente favorables, ahorrándose de esta forma generar más estados de manera innecesaria.

- Heap: la elección de esta estructura de datos para guardar los estados abiertos y cerrados es fundamental, ya que esta estructura

garantiza que se pueda insertar y extraer el estado con la mejor heurística con un coste O(log(n)).

- Evitar estados repetidos: Antes de si quiera calcular el peso de los hijos de los estados, se preguntan si están en el Heap de abiertos y cerrados, en el caso de estar en el de los cerrados, si la profundidad del nuevo estado es menor al mismo estado que ya estaba en los cerrados, significa que en realidad había un camino más corto para llegar a cierto estado, por ende, se intercambian los estados, poniéndose el estado con menor profundidad en los estados cerrados. Esto último es de crucial importancia ya que ayuda a encontrar la solución con la menor cantidad de pasos.

- Algoritmo A*: Como es sabido el algoritmo A* es reconocido por su gran eficiencia, el cual, con los tres puntos anteriores, hace que el problema se solucione de manera rápida y eficiente.

En conclusión, la solución al problema es eficiente por la combinación del algoritmo A* y los puntos mencionados.

Tabla de tiempos de ejecución archivos de prueba:

Nombre Archivo	Dimensión	N° de Pasos	Tiempo de ejecución [Segundos]
tablero1.txt	1x1	0	0,000009
tablero2.txt	2x2	3	0,000029
tablero3.txt	2x2	4	0,000037
tablero4.txt	3x3	31	0,037881
tablero5.txt	3x3	21	0,001614
tablero6.txt	3x3	23	0,00085
tablero7.txt	4x4	96	0,23779
tablero8.txt	4x4	56	0,315793
tablero9.txt	4x4	72	0,163408
tablero10.txt	5x5	165	0,175514
tablero11.txt	5x5	193	14,1459
tablerobonus.txt	5x5	62	0,212349

Ejecución del Código

Para poder ejecutar de manera correcta el programa, se necesita operar en un entorno con el sistema operativo, para prevenir errores o fallos, este programa fue escrito y probado en Ubuntu 23.10. Además, debe asegurarse de que todos los archivos .h y .cpp se encuentren en un mismo directorio/carpeta, así mismo los archivos .txt de los tableros deben estar organizados en el mismo directorio. A continuación, se describe el proceso que se debe seguir para la correcta ejecución del programa:

1. Abrir la terminal:
    - Abra una terminal en Linux y navegue hasta el directorio que contiene todos los archivos mencionados con anterioridad. Si quiere ahorrarse el paso de ir navegando hasta llegar al directorio, simplemente abra este y ejecute una nueva terminal.
2. Compilación [makefile]:
    - Ejecute el comando make en la terminal para que automáticamente se compilen los archivos necesarios para la correcta ejecución del programa.
3. Ejecución:
    - En la misma terminal, si desea ejecutar el programa introduzca “./main”, esto hará que el programa main se ejecute.
    - El programa pedirá a usted que introduzca el nombre del archivo en el cual se encuentra el tablero, en el directorio ya se encontraran algunos archivos, por ende, usted puede introducir por ejemplo “tablero4.txt” y el programa se ejecutara para encontrar la solución a dicho tablero.
    - Importante: si los archivos no se encuentran en el mismo directorio es posible que la compilación del programa no se realice con éxito, a su vez si el archivo que contiene la matriz se escribe mal, se pedirá al usuario introducir correctamente el nombre
4. Resultados esperados:
    - Si todos los pasos resultaron éxitos, entonces el programa imprimirá los pasos que se deben seguir para resolver el problema y el tiempo que tomo el algoritmo en encontrar la solución a este mismo.
    - Si el tablero no tenía solución, se imprimirá un mensaje que expresara que no tiene solución el tablero, a su vez si el archivo tiene una matriz que no representa un tablero tradicional del puzzle (por ejemplo, que tenga números repetidos), se le hará saber al usuario.

Bibliografía

- *Michael Kim | Solving the 15 Puzzle*. (2019, 27 enero). https://michael.kim/blog/puzzle
- Hasan, D. O., Aladdin, A. M., Talabani, H. S., Rashid, T. A., & Mirjalili, S. (2023). The Fifteen Puzzle—A New Approach through Hybridizing Three Heuristics Methods. *Computers*, *12*(1), 11. https://doi.org/10.3390/computers12010011
- *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/