Evaluación 1 - Taller de Programación
generalización juego del 15

Prof: Pablo Román

Marzo 18 2024

1 Objetivo

Desarrollar una aplicación eficiente y heurística del algoritmo A* para resolver problemas de búsqueda de soluciones para un determinado problema. Se implementará en el lenguaje C++, el uso de makefile, la estructuración y buenas prácticas de un código orientado al objeto. Adicionalmente se deberá construir al menos una estructura de datos que permita obtener eficiencia en la resolución del problema particular. Dicha estructura debe estar basada en arreglos.

2 Algoritmo A*

Este algoritmo se verá en detalle en clases. Este tipo de procedimientos se utilizan cuando la respuesta a un problema determinado corresponde una secuencia de pasos a realizar (`https://en.wikipedia.org/wiki/A*_search_algorithm`). El algoritmo se basa en la noción de estado, este contiene el valor de cada variable del problema. Entonces los pasos corresponden a cambios en los valores de las variables de dicho estado. Esto conforma un grafo, donde cada nodo es un estado con valores particulares de dichas variables y los pasos a conexiones dirigidas entre nodos. Adicionalmente existe un estado de inicio del sistema y una condición para el estado final. La solución entonces es buscar un camino entre el estado inicial y alguna solución final.

El algoritmos A* va construyendo el grafo a medida que se va ejecutando y se asegura de no repetir nodos ya visitados. Para ello mantiene una estructura de datos para los nodos por visitar y los ya visitados.

El algoritmo A* permite resolver combinaciones de operaciones para llegar a cierto resultado. En la práctica dicho algoritmo se utiliza cuando se dispone de un conjunto de operaciones que cambian el estado del sistema. En ingeniería existe una infinidad de aplicaciones en la cuales se utiliza. Por ejemplo, el caso de un robot autónomo que requiere de llegar a cierto destino dado un mapa donde las operaciones son pasos discretos, en seguridad informática donde se desea testear si se puede romper las barreras de seguridad se busca cierta secuencia operaciones que vulneran el sistema, para ciertas empresas se busca encontrar la secuencia de operaciones de compra/venta que permitan lograr cierto objetivo. Otra aplicación clásica es resolver el cubo rubik.

3 Problema a resolver: Puzzle 15 generalizado

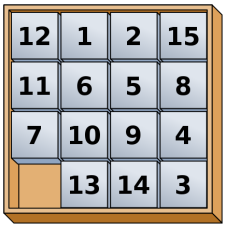

Figure 1: Una configuracion del puzle 15 (Fuente: Wikipedia)

El puzzle 15 (`https://en.wikipedia.org/wiki/15_Puzzle`) es un tablero de 4x4 donde hay dispuestos fichas con números del 1 al 15. Existe una casilla que está vacía y permite mover las fichas vecinas en el. Dado estos

movimientos el objetivo del juego es reordenar el tablero de tal forma que las fichas se encuentren en orden numérico de izquierda a derecha y de arriba hacia abajo. Es decir de la siguiente forma:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Este problema ha sido estudiado en varios papers de computación combinatorial , (`https://www.cs.cmu.edu/afs/cs/academic/class/15859-f01/www/notes/15-puzzle.pdf`, `https://link.springer.com/content/pdf/10.3758/BF03193214.pdf`, `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6883190`,`\url{https://www.mdpi.com/2073-431X/12/1/11}`, `https://cahlik.net/preprints/2021-near-optimal-solving.pdf`). **El problema generalizado es cuando el tablero es de nxn**. En dicho caso se tienen n^2-1 fichas de números. Encontrar alguna serie de pasos que logre resolver este puzzle es considerado un problema sencillo de resolver mediante A*. Sin embargo, al momento de encontrar el camino mas corto posible este problema es conocido por ser de orden no polinomial en el numero de fichas. Para esta evaluación se requiere implementar mediante el algoritmo A* un revolvedor de dicho juego generalizado en el menor numero de pasos. Es decir, debe recibir la configuración inicial y retornar la serie de pasos a realizar para resolverlo.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

4 Implementación

Se requiere implementar un programa que resuelva el juego generalizado. Se debe cuidar la orientación al objeto: No deben haber funciones sueltas, solo clases y funciones main para test y programa principal. Para ello será importante la forma en que se representará el estado del sistema y los pasos a realizar. Es importante que el programa final deberá resolver en forma eficiente el problema. En esta tarea NO puede utilizar librerías de STL, debe construir uds mismo los tipos de datos necesarios. Debe incluir una interfaz de usuario que permita indicar el nombre del archivo con la matriz de números en desorden. Estos números vienen separados por un espacio Los archivos de entrada pueden ser:

12	1	2	15
11	6	5	8
7	10	9	4
0	13	14	3

Donde el 0 indica el espacio vacío. Otro ejemplo:

1	2	3
0	4	6
7	5	8

La salida a entregar es imprimir las operaciones a realizar o pasos necesarios para ordenar los números. Se requiere que la cantidad de pasos sea la menor posible y que el programa sea rápido. Debe funcionar para los casos n=1,2,3,4,5; aunque en el último caso puede demorarse.

Además el entregable debe contener:

- Un makefile que compile programa principal y programas de test con compilación separada.
- Un test por cada clase generada. Debe comprobar que efectivamente la clase funcione.
- Varias clases implementando razonablemente la funcionalidad pedida.
- Un main.cpp con un menu para seleccionar archivo de entrada. Se resuelve el problema entregando los pasos a realizar si es que tiene solucion y el tiempo que demoró en resolverse. Si no tiene solución debe indicarlo. Se repite en un loop que incluye un salir.

- pdf de informe
- varios ejemplos de entrada. Dichos ejemplos los puede obtener de la página de Fogelman.

El programa a construir debe estar codificado en C++, implementando las funcionalidades del algoritmo A* de la manera mas eficiente posible. Además de las estructuras de datos auxiliares más **una heurística** que permita que la simplificación sea los más óptima posible. Dichas estructuras de datos auxiliares pueden ser basadas en los tipos de datos presentes en la librería STL de C++. Debe entregar una carpeta comprimida cuyo nombre corresponda a ApellidoNombre.zip (u otra compresión). Esta carpeta contiene archivos Header (.h), clases (.cpp), programas principales (main.cpp y test_XX.cpp), al menos un test por clase, un makefile con compilación separada. Toda clase debe ser implementada por separado en dos archivos (.h y .cpp). El nombre de una clase siempre comienza en Mayúscula y el archivo que la implementa tiene su nombre. No se permite definir funciones fuera de las clases y cada clase debe servir a un propósito o abstracción.

1. (10%) Informe : contiene una descripción de la estrategia de resolución que se utilizó y el porqué su implementación es eficiente.
2. (30%) Código : se revisa si compila (0 pts si no compila), si se implementa el algoritmo propuesto, si tiene la estructura indicada (clases+test+main+makefile), si se efectúa compilación separada, si esta todo descrito por clases y no hay funciones sueltas salvo la función main, si el makefile opera bien, si el código se entiende (comentarios, variables con nombres descriptivos, identación, sin repeticiones forzadas), si se encuentra 1 test adecuado por clase, si el programa se ejecuta sin problemas y efectúa las operaciones que se espera de la tarea (si no ejecuta en ningún caso 0 pts). **Si tiene funciones fuera de las clases tiene 0 puntos.**
3. (50%) Programa correcto y Eficiencia (0 pts si no funciona.): Se revisa que no existan errores de lógica y/o ejecución, que entregue los resultados correctos en todos los casos (10%). Se revisa que se haya implementado el problema de manera eficiente (40%, 0 si no hay heurística o estructuras de datos eficientes).
4. (10%) Revisión por pares. Cada compañero revisa el código a otro con una nota. Se revisará dicha revisión con otra nota. Si la revisión tiene defectos notables, la nota anterior se reemplaza por otra nota indicada por el profesor. La nota final es la nota propuesta por un par (si es que está correcta) promediada con la evaluación de la revisión que realizó el alumno.
5. (+ 1pto) Bonus al programa que genera la solución en la menor cantidad de pasos posibles. Si hay varios con la menor cantidad de pasos, entonces se selecciona al programa mas rápido.

5 Entrega

8 de abril a las 11:55PM entrega que incluye lo anterior. 1 punto por día de atraso considerando entrega hora/fecha posterior a la indicada. 12 de abril entrega de la revisión de a pares. El archivo entregable es una carpeta con nombre-rut del alumno y comprimida en un archivo con nombre-rut con los archivos requeridos.

vía Classroom