Evaluación 1 - Taller de Programación
generalización del problema de los bidones de agua

Prof: Pablo Román

Septiembre 24 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 estructuras de datos que permita obtener mayor eficiencia en la resolución del problema particular. Dicha estructura debe estar basadas 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 o el puzzle 8.

3 Problema a resolver: N recipientes de agua

Figure 1: Caso N=3 recipientes, y el objetivo es generar 6,4,0 litros

El caso original $N = 2$ (`https://www.iaeng.org/publication/IMECS2015/IMECS2015_pp138-140.pdf`) se tienen dos recipientes A_0 de 3 y A_1 5 lts de capacidad. Inicialmente dichos recipientes se encuentran vacíos y ellos se pueden llenar y vaciar de agua. Adicionalmente, se puede trasvasar el contenido de un recipiente a otros sin

botar agua. Es decir si por ejemplo en A_1 se dispone de 5 lts de agua y la otra esta vacía $A_0 = 0$, la operación de trasvasar el agua deja $A_0 = 3$ y $A_1 = 2$. En resumen se pueden realizar 6 operaciones:

Operacion	*Descripcion*
O_0	$VaciarA_0$
O_1	$VaciarA_1$
O_2	$LlenarA_0$
O_3	$LlenarA_1$
O_4	$TrasvasarA_0 \rightarrow A_1$
O_5	$TrasvasarA_1 \rightarrow A_0$

El juego se resuelve conociendo una combinación de operaciones que lleva desde los recipientes vacíos a cualquier configuración donde $A_1 = 4$. Por ejemplo la secuencia:

Paso	*Operacion*	A_0	A_1
0		0	0
1	O_3	0	5
2	O_5	3	2
3	O_0	0	2
4	O_5	2	0
5	O_3	2	5
6	O_5	3	4

Este problema (`https://en.wikipedia.org/wiki/Water_pouring_puzzle`) ha sido estudiado en varios papers de computación combinatorial, (`https://www.davidpublisher.com/Public/uploads/Contribute/55262b60220ec.pdf`, `https://www.parabola.unsw.edu.au/sites/default/files/2024-03/vol52_no1_3.pdf`, `https://www.iaeng.org/publication/IMECS2015/IMECS2015_pp138-140.pdf`).

Se puede generalizar el problema de los dos recipientes de agua a varios recipientes. En dicho caso se tienen N recipientes $\{A_i\}$. Cada recipente A_i tiene capacidad máxima de C_i. Notar que los números C_i deben ser primos entre ellos para poder lograr cualquier secuencia posterior. Se dispone de las mismas operaciones de intercambio de agua pero entre pares (i, j) de recipientes y de vaciado/llenado para cualquier recipiente i. El objetivo del problema es encontrar una secuencia de operaciones que logre concretar valores $\{V_0, V_1, ..., V_{N-1}\}$ de agua en los recipientes. En resumen las entradas del algoritmos son:

- N: Cantidad de recipientes
- $C = [C_0, ..., C_{N-1}]$: Arreglo de capacidades de cada recipiente.
- $V = [V_0, ..., V_{N-1}]$: Arreglo de valores a obtener en cada recipiente.

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 la cantidad mas corta posible de operaciones para este problema es conocido por ser de orden no polinomial en N.

Para esta evaluación se requiere implementar mediante el algoritmo A* un programa que resuelve dicho juego generalizado en el menor número de operaciones posible. Es decir, debe recibir la configuración inicial y retornar la serie de pasos a realizar para resolverlo o indicar que no puede ser resuelto. **El programa deberá ser lo más eficiente posible en tiempo de ejecución.** Se conoce que la búsqueda en anchura efectivamente encuentra la secuencia de operaciones mas corta. Sin embargo, como se ha visto en clases genera ineficiencias al transformarse en una búsqueda exhaustiva en las soluciones. En este sentido su implementación deberá mejorar la búsqueda en anchura a una búsqueda heurística para intentar mejorar los tiempos y al mismo tiempo minimizar el número de operaciones.

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 **las operaciones** a realizar. Es importante que

el programa final deberá resolver en forma eficiente el problema. **En esta tarea NO puede utilizar librerías de STL que implementen tipos de datos eficientes**, debe construir ud mismo todos los tipos de datos necesarios. En particular debe basarse en arreglos y NO utilizar la clase vector. Otra restricción de la implementación es que se debe evitar un listado de instrucciones IF. Para ello se siguiere por ejemplo crear una clase Operacion la cual implementa un método operar sobrecargado con uno y dos argumentos. Esto permite automatizar el barrido de todas las operaciones posibles.

Debe incluir una interfaz de usuario que permita indicar el nombre del archivo con la configuración inicial.

El programa recibe como entrada los arreglos C y V deduciendo el valor N. Deberá verificar el caso que son incompatibles o esta mal conformado. Debe cumplir $V_i \leq C_i$.

Como ejemplo de archivo de entrada podemos considerar el problema original:

$$\begin{matrix} 3 & 5 \\ 0 & 4 \end{matrix}$$

La salida a entregar es imprimir las operaciones a realizar o pasos necesarios para resolver el problema. Se requiere que la cantidad de pasos sea la menor posible y que el programa sea rápido. Debe funcionar para los casos que se entregarán en clases.

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.
- Cada clase son 2 archivos (.cpp) y (.h), que deben ser compilados en forma separada.
- 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.

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 sólo propósito o abstracción.

1. (10%) Informe : contiene una descripción de la estrategia de resolución que se utilizó y el porqué su implementación es eficiente y encuentra secuencias cortas de operaciones.
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 o no tiene ningún intento de hacerlo eficiente.): 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 y 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 (debe ser eficiente además). Si hay varios con la menor cantidad de pasos, entonces se selecciona al programa mas rápido.

5 Entrega

21 de Octubre a las 11:55PM entrega que incluye lo anterior. 1 punto por día de atraso considerando entrega hora/fecha posterior a la indicada. 23 de Octubre 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