TALLER N°1

Evaluación 1 - Taller de Programación

Cruzando el río

usach

Taller de programación 1-2023

Fecha: 06/04/2023

Autor: Lucas Mesías

TALLER N°1

Cruzando el río

Explicación breve del algoritmo

El problema del río se trata de un conjunto de números, cada número representa un objeto, todos los objetos comienzan a la izquierda del río, y queremos moverlos todos a la derecha usando un bote de capacidad limitada, además algunos objetos son conductores, y debe haber al menos 1 conductor en el bote para que sea válido moverlo. Entonces, queremos buscar los pasos que nos llevan a dicha solución, así que representamos cada "paso" como un estado, cada estado estará conectado con el paso anterior, y con el algoritmo vamos creando los "caminos" hasta llegar a la solución. De esta forma al llegar al estado final, podemos ver el camino recorrido.

Heurísticas o técnicas utilizadas

El algoritmo a utilizar es el A*, una generalización del algoritmo de dijkstra. En este, tendremos un conjunto de estados abiertos y cerrados, con un ciclo que se repetirá hasta encontrar el estado final o hasta que se nos acaben los estados abiertos. En cada iteración se saca del conjunto abierto un estado y se inserta en el conjunto cerrado, luego se generan los estados que se pueden crear a partir de este, y si el estado generado es nuevo (es decir, no se encuentra en los conjuntos abierto y cerrado), se agrega al conjunto abierto.

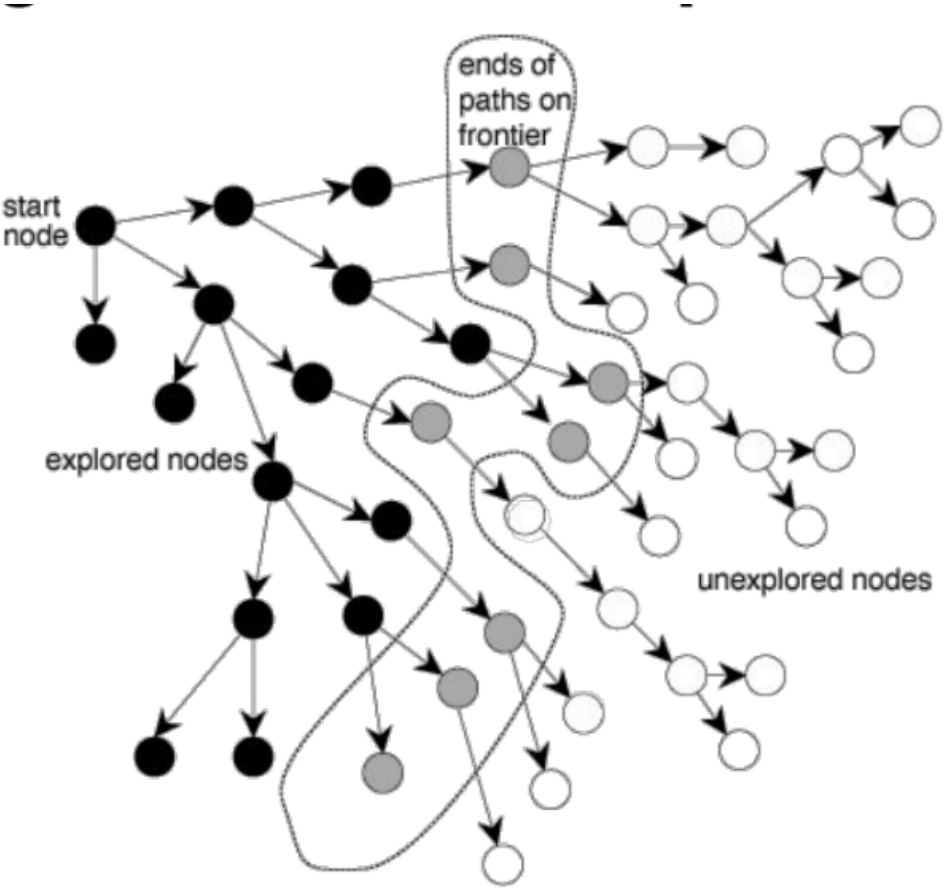
De esta forma se recorren los caminos nuevos y se evita la repetición o redundancia de caminos.

Funcionamiento del programa

Lo primero que se crea son los objetos AVL que contendrán los conjuntos abiertos y cerrados, esto sucede en el constructor de la clase RiverCrossing, luego al llamar a la función solve se comienza a ejecutar la sección principal del programa, primero se lee el archivo que contiene la información del problema y se generan las operaciones, todo esto dentro de la función getProblemInfo. Luego se crea un estado inicial con todos los objetos a la izquierda del río y se inserta en el conjunto abierto, y se comienza el loop principal del algoritmo: while (!this->openAVL->isEmpty())

Dentro de este loop, se saca del AVL un elemento con la función pop(), siendo que un AVL no tiene un criterio definido para el método pop, se extrae el elemento más grande del árbol. Se revisa que el estado recién obtenido sea el estado final, en caso de serlo se imprime el camino recorrido y se termina el programa, en caso contrario se introduce el estado en el conjunto cerrado, y se aplican todas las operaciones generadas a este estado, insertando los

movimientos válidos al conjunto abierto, comenzando el loop hasta que el conjunto abierto sea vacío, significando en ese caso que no existe una solución.



Orientación al objeto en C++ - Pablo Román

En el gráfico anterior se muestran los nodos del conjunto cerrado en color negro y los nodos del conjunto abierto en color gris.

Aspectos de implementación y eficiencia

Para mejorar la eficiencia del código se implementaron varios cambios con respecto a la versión original del algoritmo entregada por el profesor, los conjuntos abiertos y cerrados se cambiaron por un AVL, el cual tiene una mejor combinación de complejidad de búsqueda, inserción y eliminación, esto es importante porque estos métodos se llaman en cada iteración.

El segundo cambio importante fue la eliminación del arreglo de booleanos por un número entero sin signo, el cual se trabaja de la misma forma, cada bit representa si un objeto está presente en el lado correspondiente, en este caso, el derecho. De esta forma, las operaciones de comparación de restricciones pasan de ser una comparación entre arreglos y matrices, a ser una comparación de un número entero sin signo y un arreglo de números enteros sin signo, además se pueden usar operadores lógicos para comprobar la validez de movimientos, y realizar dichos movimientos, sin hacer uso de loops para ninguna de estas

operaciones. Aunque este acercamiento reduce el número de items con los que podemos trabajar a la cantidad de bits que puede contener un entero sin signo (32 en c++), y aunque se puede cambiar el tipo de unsigned int a unsigned long long u otros, esta limitación aparecerá eventualmente, y se debe cambiar el código.

Los últimos cambios fueron ordenar el arreglo de operaciones según la cantidad de 1s que contienen, de esta forma, se priorizan los movimientos que mueven más items a la vez. Y similarmente, el método pop() que elige el nodo a sacar del conjunto abierto saca el estado con el mayor valor del entero sin signo, ya que el estado final será siempre el número más grande de todos (en representación de arreglo de booleanos, este estado contendría solo 1s), de esta forma, si el estado final se encuentra entre los abiertos, se priorizaría abrir ese estado primero, o en otro caso, se eligen estados con mayor probabilidad de tener más 1s ya a la derecha del río, aunque este no es siempre el caso.

A la hora de generar las operaciones se utilizó una función recursiva, que hace posible evitar generar las 2^n operaciones posibles, creando sólo las combinaciones posibles que caben en el bote.

Ejecución del código

En el proyecto se utilizaron las librerías *iostream*, *fstream*, *string* y *sstream* para obtener datos por consola, el nombre del archivo a leer y realizar la lectura del archivo del problema, estas librerías fueron incluídas por el profesor, en el código que nos entregó de base. Luego, se incluyó la librería *vector*, para almacenar las operaciones en un arreglo, y aunque se pretendía usar la función *sort()* incluída, se terminó implementando una función propia, ya que se requería ordenar el arreglo según un criterio propio.

Para ejecutar el código, se descomprime el archivo zip, ya sea usando click derecho y la opción de extraer, o con el comando 'unzip LucasMesias212666599', luego se abre una terminal dentro de la carpeta recién extraída, se usa el comando 'make' seguido de la prueba que se quiera ejecutar, finalmente se ejecuta el programa con el comando correspondiente (./'testDeseada').

- make test (Prueba principal)
- make testAVL
- make testFileReader
- make testNode
- make testOperation
- make testState

Para ejecutar la prueba principal, el archivo de lectura con el problema debe estar en la carpeta principal que contiene el código fuente.

Bibliografía

Roman, P. (s.f.). Orientación al objeto en C++ [Presentación en PowerPoint]. Archivo PowerPoint. Recuperado de https://drive.google.com/file/d/1NIeDhTmCsF9fomXe3zLBLGepynXes8jP/view