Práctica temas 2-3 – AEDNavigator

AED I – 2022/2023

Componentes:

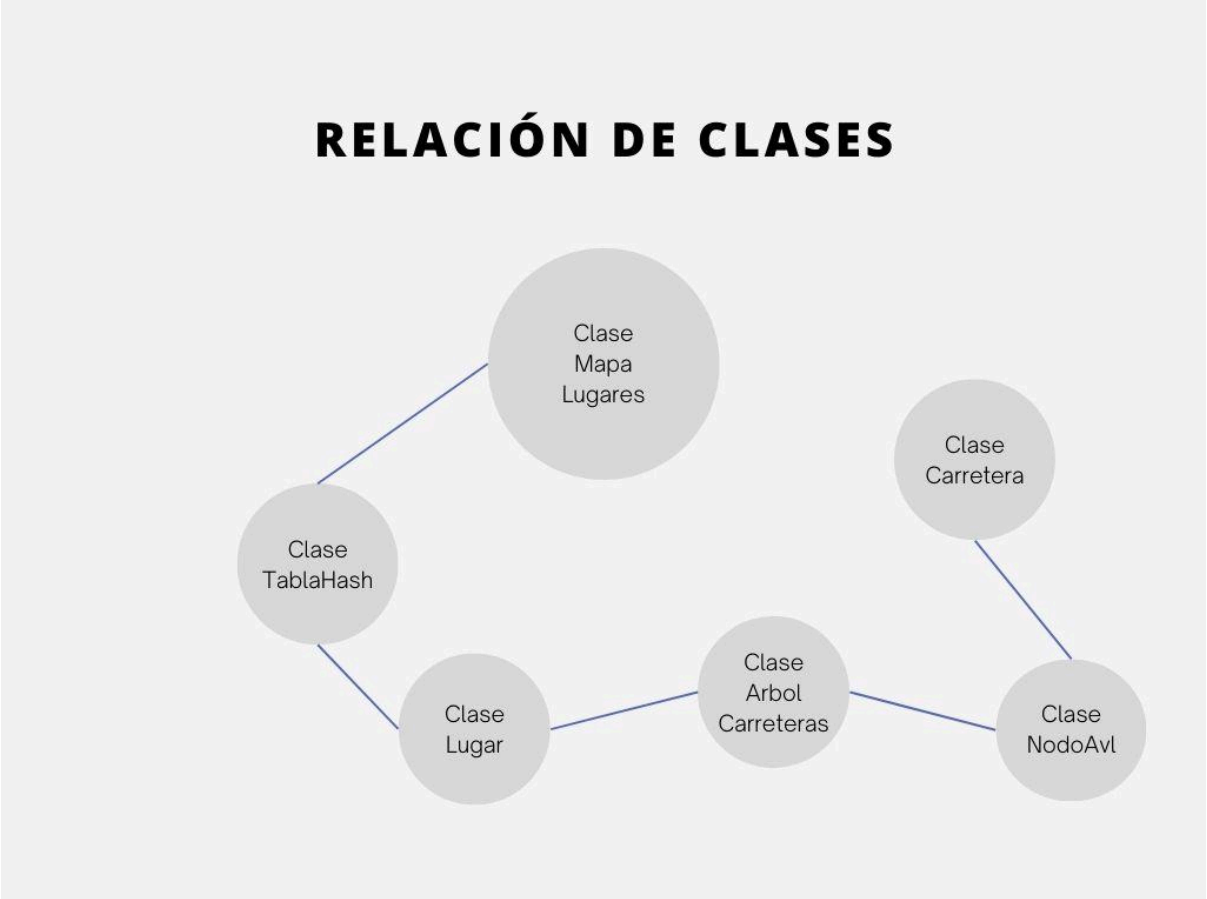
Sergio Ros Liarte

José María López Serrano

Índice

Análisis y diseño del programa

Clases definidas



Clase Lugar:
TAD que representa un lugar, paraje o población (por ejemplo la ciudad de Murcia), se almacena su nombre, una cadena de información de cualquier tipo, y una estructura de datos arborescente que guarda las carreteras que conectan el lugar.

Clase Carretera:
TAD que representa un camino o carretera entre dos TAD Lugar. tienen dirección (si se introduce, por ejemplo, una Carretera Murcia-Cartagena, no se incluirá Cartagena-Murcia). Almacena un puntero al Lugar destino, un coste que representa la longitud del camino y una cadena de información. El Lugar origen no se almacena porque cada Lugar almacena las Carreteras de las que es origen en una estructura arborescente.

Clase ArbolCarreteras:
Estructura arborescente asociada a cada Lugar. Almacena las carreteras cuyo origen sea el lugar al que pertenece el árbol. Está formado por NodosAvl.

Clase NodoAvl:
Estructura de los nodos a partir de la cual se construyen los Árboles de Carreteras. contienen una cadena (clave) que los representa, y que a su vez es el nombre del Lugar destino de la Carretera que almacenan, una Carretera, su altura (para la representación del árbol) y dos punteros, uno a su hijo izquierdo y otro a su hijo derecho, pues el árbol seguirá la estructura de un árbol binario (con condición de balanceo).

Clase TablaHash:
Tabla de dispersión que almacena los Lugares que se vayan insertando. Es accedida por MapaLugares (que la incluye como atributo) para consultar e insertar Carreteras y Lugares.

Clase MapaLugares:
MapaLugares representa la estructura de datos para almacenar Lugares. Consta de una TablaHash de Lugar a la que accede para consultar o insertar Lugares, y a partir de ella accede a los propios Lugares para insertar o consultar carreteras en sus correspondientes ArbolesCarreteras.

Módulos creados

Main (302.cpp):
Programa principal, incluye también el intérprete de comandos que recoge la entrada de la consola (o del fichero de redirección) y lo convierte en comandos que entiende el MapaLugares.

En este módulo se incluyen Lugar.h, Carretera.h y MapaLugares.h

Carretera(Carretera.cpp y Carretera.h):
Módulo que implementa la clase Carretera y sus métodos disponibles.

El fichero de cabecera (.h) no incluye ningún otro módulo, sin embargo hace una declaración adelantada de la clase Lugar para definir el atributo destino.
El fichero de implementación (.cpp) incluye Lugar.h (y el propio Carretera.h para compilarse).

Lugar(Lugar.cpp y Lugar.h):
Módulo que implementa la clase Lugar y sus métodos disponibles.

El fichero de cabecera (.h) incluye Carretera.h y ArbolCarreteras.h porque la clase Lugar declara un atributo de clase ArbolCarreteras y contiene métodos que devuelven una Carretera.

El fichero de implementación (.cpp) no incluye nada adicional aparte de Lugar.h porque todo lo que necesita se incluye en Lugar.h.

TablaHash(TablaHash.cpp y TablaHash.h):
Módulo que implementa la clase TablaHash y sus métodos disponibles.

En el fichero de cabecera (.h) incluye Lugar.h para definir el tipo de datos que se almacena en las cubetas de la tabla.
En el fichero de implementación (.cpp) no incluye nada además del propio fichero de cabecera porque la Tabla solo maneja Lugares.

ArbolCarreteras(Arbolcarreteras.cpp y ArbolCarreteras.h):
Módulo que implementa la clase ArbolCarreteras y sus métodos disponibles.

El fichero de cabecera (.h) incluye NodoAvl.h (porque el árbol se construye a partir de nodos individuales) y Carretera.h porque tiene métodos que insertan o devuelven Carreteras.
El fichero de implementación (.cpp) incluye, además de ArbolCarreteras.h, Lugar.h porque ciertos métodos necesitan acceder a métodos de la clase Lugar.

NodoAvl(NodoAvl.cpp y NodoAvl.h):
Módulo que implementa la clase NodoAvl y sus métodos disponibles.

El fichero de cabecera (.h) incluye solo Carretera.h (para implementar el atributo del Nodo que guarda una carretera).
El fichero de implementación (.cpp), al igual que el de ArbolCarreteras, incluye NodoAvl.h y Lugar.h, porque ciertos métodos necesitan acceder a métodos de la clase Lugar.

Makefile
Contiene todas las dependencias entre módulos así como los comandos para compilar conjuntamente el código y enlazarlo en un ejecutable .out.

Tabla de dispersión de Lugares

Para implementar el tipo TablaHash hemos seguido una estructura de dispersión abierta. Aparte de tener una mayor simplicidad a la hora de implementarla, pues no es necesario definir una función de redispersión, creemos que es una mejor opción porque, comparando con una dispersión cerrada, a mayor porcentaje de llenado de la tabla, la secuencia de búsqueda de un Lugar en la tabla de dispersión abierta es lineal mientras que en una de dispersión cerrada sería exponencial, por tanto habría que reestructurar la tabla muchas más veces, y esta es una operación bastante costosa (Es cierto que si no se reestructura, la tabla de dispersión abierta puede ser todavía más ineficiente aumentando considerablemente el tiempo de búsqueda, pero implementamos reestructuración así que descartamos ese problema) . Para implementarla creamos un array dinámico de listas de TAD Lugares (implementadas en la librería <list> de c++) . La simplicidad sintáctica también nos decantó por esta opción pues <list> contiene todo tipo de funciones ya definidas para insertar, buscar elementos y eliminar, así como iteradores, quedando por implementar solo lo referente a la propia dispersión y la elección de cubetas.

Como función de dispersión, y dado que las claves que teníamos eran nombres de lugares, había que trabajar con caracteres, por lo que era necesario trabajar con una función de dispersión con secuencias. Nos decantamos por un método de suma exponencial. En un principio usamos base 10: h(k) = x1 + x2*10 + x3*100 + x4*1000 … Pero dado que 10 no es primo creímos mejor cambiarlo por uno que sí lo fuera y la final es en base 7.

$h(k) = \Sigma(Xi * 7^{(i-1)}) \text{ MOD } M$

Es una función rápida en cuanto a cálculos y genera una buena dispersión, si bien quizás se podría mejorar por ejemplo el método de la multiplicación al sumatorio, aunque nosotros preferimos usar una función simple.

La tabla tiene implementada reestructuración. El funcionamiento es el siguiente: MapaLugares lleva la cuenta de los lugares que se insertan. Al llegar la tabla a nº lugares >= 1.5* nº de cubetas existentes se realiza la reestructuración. El algoritmo es:

1. Copia la tabla a un apuntador auxiliar
2. Coloca en el apuntador de la tabla de MapaLugares una nueva tabla vacía con el doble de cubetas de la anterior.
3. Resetea (a 0) el número de Lugares guardados (puesto que al insertarlos de nuevo volverán a contarse)
4. Recorre la tabla antigua guardada en auxiliar y para cada elemento lo inserta en la nueva.
5. Elimina la tabla antigua.

El destructor de la tabla se apoya en la función vacia() de la propia clase TablaHash. El método vacía recorre la tabla, obteniendo los punteros a Lugares de los que se compone, ejecuta el destructor de Lugar sobre esos punteros y una vez vacía ejecuta el método clear() de <list> que vacía la lista de los punteros inservibles que se han quedado. También se puede realizar la función vacia() por separado, y es justo lo que hace MapaLugares cuando recibe la orden Inicializar del intérprete de comandos del programa principal.

Árboles de carreteras

A la hora de implementar los árboles de carreteras barajamos 2 posibles opciones, árboles de prefijos (trie) o árboles AVL (descartamos los árboles B porque no los dominamos lo suficiente y nos parecían demasiado complicados de implementar, aunque seguramente hubieran sido bastante óptimos). Al final nos decidimos por AVL porque los trie requerían una implementación de nodo demasiado compleja (necesitábamos declararlos con punteros genéricos a hijos para poder almacenar carreteras, tratar especialmente los nodos hoja, etcétera.) El árbol AVL nos proporcionaba una implementación no demasiado compleja y la la vez presentaba ventajas, especialmente a la hora de hacer operaciones como ListarAdyacentes o ConsultarCarretera, donde el balanceo y, usando comparación lexicográfica para las cadenas de destino de las Carreteras como clave nos era muy útil.

Los nodos del árbol (clase NodoAvl) tienen como atributos: un objeto Carretera, una clave (el nombre del destino de Carretera), un entero que representa su altura y dos punteros a NodoAvl, hijoDer (su hijo derecho) e hijoIzq (su hijo izquierdo). El árbol simplemente contiene un apuntador a un NodoAvl denominado raíz, que representa a la raíz del árbol. La raíz se inicializa a NULL, así como los hijos de un NodoAvl.

Los adyacentes de un Lugar se listan de manera recursiva. Primero se busca en la raíz de su árbol de carreteras. Si esta es NULL no se devuelve nada, si no es así se devuelve una cadena compuesta por la operación recursiva sobre el hijo izquierdo + la clave del NodoAvl actual + <, > (coma separadora) + la operación recursiva sobre el hijo derecho. Al final se elimina el <, > sobrante si lo hay.

La condición de balanceo del árbol se comprueba cada vez que se inserta una nueva Carretera. Primero se comprueba si existe esa carretera y sino existe se introduce en un nodo que esté a NULL, en la posición que le corresponda (el procedimiento inserta busca recursivamente el lugar donde hay que insertarla comparando en orden alfabético las cadenas, para mantener la condición de que, en un nodo, su hijo izquierdo debe ser menor que él, y su hijo derecho mayor, y actualiza la altura del nodo). Después comprueba la diferencia de altura entre los hijos del nodo y según se haya insertado en el hijo derecho o izquierdo y según el caso (II, ID, DI, DD) se realiza una rotación u otra (II-RSI, DI-RDI, ID-RSD, DD-RDD).

La liberación del árbol se realiza de forma recursiva. El destructor del árbol (que se ejecuta cuando se destruye el Lugar que lo contiene) ejecuta delete raiz, y el destructor de NodoAvl se encarga de llamarse recursivamente sobre hijoIzq e hijoDer hasta llegar a los nodos hoja NULL, donde se detiene la recursividad. El destructor no ejecuta nada más pues el resto de sus atributos son estáticos y se eliminan automáticamente sin necesidad de especificarlo explícitamente en el destructor.

Las carreteras que se almacenan en los nodos del ArbolCarreteras referencian a Lugar con un apuntador de memoria. El apuntador se genera en la construcción de la carretera, donde hay que acceder obligatoriamente a la tabla para poder saber si el destino existe. En ese momento se enlaza el objeto al que se referencia en la tabla (la tabla también almacena punteros, así cuando se reestructura la misma no se pierden los punteros del árbol), y ya no es necesario acceder dos veces a la tabla de dispersión para consultar una carretera.

Variables globales

Existe una variable global en el programa principal que es del tipo MapaLugares.