Report C++ Assignment #2

Pair B3408
Renault Benoit, Espeute Clément

1 <Sensor> class specifications

1.1 Global presentation

Stores the ID of a given Sensor (which is a unique integer specific to each Sensor) and a tri-dimensional array (abb. "index") of Stats structures pointers, the first dimension reprensenting the day of the week (abb. "d7" or "D7"), an integer ranging from 1 to 7, the second dimension representing the hour, an integer ranging from 0 to 23, the third one representing the minute, an integer ranging from 0 to 59. Therefore, the Stats structure (abb. "struct") corresponding to a combination of [D7][Hour][Minute] allows us to access to the number of times a given state of the traffic has happened for a given Sensor at the given [D7][Hour][Minute] combination.

1.2 Structures overview

1.2.1 Structure <Stats>

Stores positive integer counters for each of the possible traffic states ('V', 'J', 'R', or 'N') into an array of size 4.

Constructor : The default constructor sets all the counters to zero.

Overload of operator+= : Allows adding two Stats structs by adding each corresponding array element separately. It is especially useful when adding Stats of different [D7][Hour][Minute] combinations to get, say, the overall Stats of the Sensor for example.

Sum method : Adds the array elements of the Stats struct together and returns it as a double-precision float. It is used to know if any counter of the Stats struct has been incremented since it's creation, or to be able to computes the relative statistics of StatsRel.

1.2.2 Structure <StatsRel>

Stores relative statistics for each of the possible traffic states ('V', 'J', 'R', or 'N') as double-precision floats into an array of size 4.

Constructor : The default constructor converts each of the four counters of the passed in Stats struct pointer into corresponding relative statistics for each traffic state.

PrintStatsRel : Displays the array elements of the StatsRel struct one per line, rounded into percentages, in the format specified in the assignment.

1.3 Functions overview

Constructor

Default constructor takes an integer in parameter and assign it as the ID of the Sensor. It then dynamically allocates the memory required for the tri-dimensional index of Stats structs pointers.

AddEvent

Takes integer parameters for the D7, hour and minute[1] and a character parameter describing the state of the traffic ('V', 'J', 'R', or 'N') of the event to be added and increments the suitable Stats struct array element in the tri-dimensional array for a given day of week, hour and minute by one.

GetStatsByMin

Takes integer parameters for the D7, hour and minute[1] and returns the corresponding Stats struct pointer from the index array.

AddStatsByHour

Takes integer parameters for the D7 and hour[1], and a pointer to a Stats struct. Adds all the Stats structs in the index matching the given day and hour into the Stats struct which pointer is passed in.

AddStatsByDay

Takes integer parameters for the D7[1], and a pointer to a Stats struct. Adds all the Stats structs in the index matching the given day into the Stats struct which pointer is passed in.

AddStatsBySensor

Takes a pointer to a Stats struct as parameter. Adds all the Stats structs in the index of the current Sensor into the Stats struct which pointer is passed in.

GetDuration

Takes integer parameters for the D7, hour and minute[1]. Computes and returns the most probable duration of the journey through the road segment associated with the Sensor at the specified [D7][Hour][Minute] time combination. Each state is associated with a duration : (V→1, J→2, R→4, N→10). If no data is available for the given moment, all the Stats counters are equal to zero, therefore affecting 1 minute ('V' case).

GetID

Getter for the Sensor ID which returns the said integer value.

PrintSensorStatsRel

Computes and displays the statistics of the current Sensor according to the format given in the assignment.

[1] Respecting the conditions mentioned at the beginning.

2 Storing the Sensors

We chose to use a BinaryTree to store the Sensors. It allows us to have a very fast indexing and insertion speed. Here are some explanations about the vocabulary that we will use later :

Node Represents an element in our tree. Each Node has two children and one parent and contains a pointer to a Sensor. The Sensor's ID determines the position of the Node in the tree, as the left child Sensor's ID is always lower than the one of it's parents, and the right one is higher.

Height The height is defined as the longest chain of Nodes in a subtree.

Balance The balance of a Node is determined like this : $leftSubtreeHeight - rightSubtreeHeight$. It allows to know if a subtree is balanced or not. In our tree, the Node balance has to be between -1 and $+1$. If it is higher/lower, we have to perform some balancing operations.

3 <Node> Class Specifications

3.1 Global Presentation

Node is a class that represents a BinaryTree element. Again, each Node contains a Sensor pointer and three Node pointers : one for the parent Node and two for the left and right child Nodes. The Nodes are sorted with the Sensor ID : Nodes that are at the left of this Node will have their Sensor ID lower to this one, and Node to the left will have a Sensor ID higher to this one.

3.2 Function overview

Constructor

By default, the Node constructor only takes a pointer to a Sensor as an argument. You can also specify pointers to the parent Node and the children left and right Nodes (defaulted to NULL). These pointers are affected to the required attributes.

GetLeft and GetRight

Simple Getters for the left and ligth childs of the current Node, return pointers to them.

GetHeight

Returns the height of this subtree as an integer.

GetBalance

Returns the value of the balance of this subtree as an integer.

Search

Searches and returns a pointer to a Node corresponding to a Sensor that has the same ID as the integer representing the passed in ID searchID in the current subtree. If no matching Node is found, it will return a NULL pointer instead.

For better comprehension : the Node from which this function is called acts as the tree's root during a search. This function uses a recursive implementation : if this Node's Sensor ID doesn't match searchID, the function is called again on the left or right children of the Node depending if searchID is lesser or greater than the Node's Sensor ID. If the ID matches, it simply returns the pointer to the current Node, and if it cannot go any further in the tree (because there are NULL pointers), it returns NULL.

Insert

This function returns a pointer to the Node with an ID matching the passed in searchID integer value in the tree, or creates a new Node with searchID and returns it if the searchID is not found.

GetSensor

Simple getter for the Sensor, returns a pointer to the Node's Sensor.

4 <BinaryTree> Class Specifications

4.1 Global Presentation

The BinaryTree class offers a useful layer of functions that can be used to manipulate the entirety of the tree. It especially keeps track of the root (the first Node in the tree), and allows to iterate through it, and add data to a specific Sensor using a single function.

4.2 Function Overview

Search

Returns a pointer to the Sensor with the matching ID. Will return NULL if that Sensor hasn't been found.

Insert

Takes an integer for the ID of the desired sensor, characters for the D7, hour, minute, and traffic state of the event to be added as parameters. Passes this data to the required Sensor's Node, calling its Insert method, therefore ensuring that the desired Sensor will be in the tree.

GetRoot

Returns a pointer to the root Node of the BinaryTree.

4.3 Iterating trough the tree

The process of iterating trough the tree is fairly intuitive. Each time we call the Iterate function, it returns a pointer to a Node in the tree that hasn't been visited yet, until we get a null pointer, indicating that we have been through the whole tree. The InitIterate method is to be used before using Iterate to ensure its good execution.

We keep track of our position in the tree via a Stack containing the path from the current Node to the tree's root. The stack is more precisely a dynamic Node pointer array with a variable keeping the current number of pointers in the array.

To iterate, we start from the root, push the next Node to be returned in the stack, and return the current Node at the end of the method. We prioritize the left Nodes, then the right ones. If we can't go down the tree any more (only NULL pointers), we pop Nodes from the stack until we can go to a right Node again.

InitIterate

This method prepares the Tree for an incoming iteration through it. It initializes the stack using the height of the tree.

Iterate

Returns a pointer to a Node that hasn't been visited yet, or NULL if we reached the end of the tree.

5 <IoEngine> class specifications

5.1 Global presentation

Manages the user inputs on the standard input stream that are described in the assignment and displays the according data. For that, it is assigned as an attribute the pointer to the BinaryTree used to store the Sensors. For details about the commands format or meaning, please directly refer either to the assignement or the documentation of the class.

5.2 Functions overview

Constructor

The default constructor affects the passed in BinaryTree pointer to the attribute.

ReadInput

Reads the input from the standard input stream and calls for the methods associated to the user commands. Returns a boolean value of false if the command 'EXIT' has been issued, true otherwise.

HandleADD

Reads the details of the ADD command and calls the necessary function thats adds the new event to the appropriate Sensor and creates the Sensor (and the associated Node in the binary tree) if they don't exist yet[2].

HandleJAM_DH

Performs the JAM_DH operation : computes and displays the percentage of traffic jam (corresponding to a 'R' or 'N' traffic state) for each hour of a given day of the week, averaging from every Sensor[2].

HandleOPT

Performs the OPT operation : computes and displays the optimal time (which reduces the duration of the journey through the given segments) of departure and the associated minimal duration of the journey[2,3]. As it calls for the GetDuration function, same consequences apply if Sensor data is missing for given time.

HandleSTATS_C

Performs the STATS_C operation : displays the statistics for each of the traffic states attributes ('V', 'J', 'R', or 'N') of a Sensor as percentages, one by line, according to the assignement[2,3].

HandleSTATS_D7

Performs the STATS_D7 operation : computes and displays the statistics of every traffic state ('V', 'J', 'R', or 'N') of a given day of the week, averaging from every Sensor[2].

[2] If the command is valid (respects the assignment format)
[3] It is assumed that every required Sensor exists or the method will simply exit.

6 <Main> class specifications

Last, but not least, this class contains the entry point of the program. In this entry point function, the BinaryTree which will contain all the Sensors is created, and then passed in to the IoEngine, which will then loop until the user issues the 'EXIT' command. Before ending, the program will display the time used to perform in seconds. If in debug mode, (`#define DEBUG`), this function will rather execute the tests described in the dedicated report. If (`#define SERIALIZE`) is uncommented too, the program will instead create and fill a binary tree with random ID values, then print a lua table representation of this BinaryTree in the standard output.