ECE 250 - Project 3
Data Structure - Quadtree
Design Document

Harshit Manchanda, UW UserID: h2mancha

Mar 13th, 2020

1. Overview of Classes

Class 1: Node

Description:
Represents a node of the Quadtree which stores the address of its North East, North West, South West and South East child nodes along with the data such as its coordinates – longitude and latitude, population, cost of living and average salary.

Member Variables:
1. String City Name (name of the city node)
2. Longitude stored as a double (coordinates of the node)
3. Latitude stored as a double
4. Population
5. Cost of living
6. Average Net Salary
7. Node type pointer NW (pointing to its child node in the NW direction)
8. Node type pointer NE (pointing to its child node in the NE direction)
9. Node type pointer SW (pointing to its child node in the SW direction)
10. Node type pointer SE (pointing to its child node in the SE direction)

For example, a node object tree will hold the addresses of its child nodes in the NW, NE, SW and SE directions along with the values Toronto as city name, 43.66 and -79.42 as longitude and latitude, 5213000 as population, 2157 as the cost of living and 3396 as average net salary.

Member Functions (operations):
1. insert: using the conditions mentioned in the description, it finds which direction of the tree to move in to insert the new node which is sent as a parameter until it finds space using recursion.
2. search: similarly, it uses recursion to traverse in that direction of the tree where it can find the node with the coordinates sent as parameters and returns the node if found
3. max: implements Inorder traversal to find the maximum value of the attribute in the node's subtree which is sent as a parameter along with the attribute and max value variable which will hold that max value.
4. min: finds the minimum value of the attribute in the node's subtree which is sent as a parameter along with the attribute and min value variable which will hold that min value using Inorder traversal.
5. total: finds the total value of the attribute in the node's subtree which is sent as a parameter along with the attribute and total value variable which will hold that total value using Inorder traversal.
6. print: prints each and every node's city_name value using Inorder traversal with root as its parameter.
7. clear: deallocates all manually allocated nodes by following Post order traversal with root as parameter
8. returnCityName: returns the value of cityName data member for the node sent as parameter.

Class 2: QuadTree

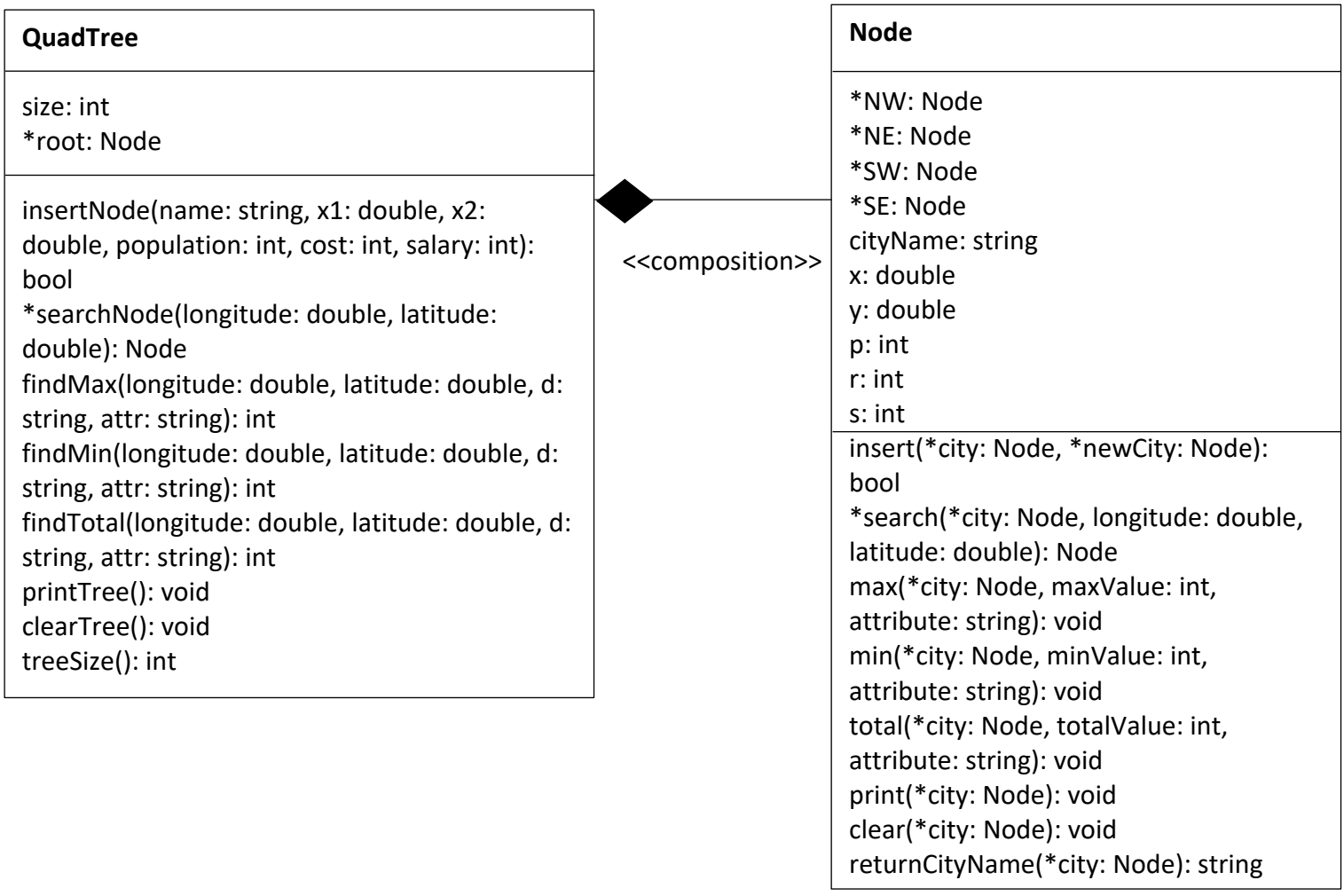
Description:
It is a tree data structure in which each internal node has 4 children, stores information about the city in the node which are objects of the Node class, allows insertion of new nodes and searches for a node if it exists in the tree or not. It has more functions like finding the max or min or total value of an attribute in the given direction, printing the tree, deleting the tree, etc.

Member Variables:
1. Size of the tree
2. Node type pointer root which points to the root of the tree

Member Functions (operations):

1. insertNode: adds the new node as root if tree is empty, otherwise calls Node's insert function with root and new city as parameters to iterate through the tree recursively to find the correct position to add the node.
2. searchNode: searches for the node with the coordinates passed as parameter by passing root and those coordinates to the Node's search function.
3. findMax: finds the max value of the attribute in the specified direction by first locating the node and then calling Node's max function with the directional node, max value and attribute passed as parameters
4. findMin: finds the min value of the attribute in the specified direction by first locating the node and then calling Node's min function with the directional node, min value and attribute passed as parameters
5. findTotal: finds the total value of the attribute in the specified direction by first locating the node and then calling Node's total function with the directional node, total value and attribute passed as parameters
6. printTree: prints the entire tree if not empty by displaying the city names using Node's print function which implements Inorder traversal
7. clearTree: clears the entire tree if not empty using Node's clear function which uses Post-order traversal to clear each node.
8. treeSize: returns the current size of the tree

UML Class Diagram -



2. Constructors/Destructors

Class Node (Constructor):
The constructor for this class assigns city name as empty string, longitude and latitude as 0.0 and population, cost of living and salary as 0 if no values are passed otherwise it assigns the values sent to the constrictor while creating the new node.

Class Node (Destructor):
The QuadTree's destructor automatically deallocates memory at the end so we don't need to have Node's own destructor.

Class QuadTree (Constructor):
There is no need for a constructor as for a new node, Node class's constructor is called and QuadTree's data members are always already assigned nullptr for root and 0 for size.

Class QuadTree (Destructor):
Since there is a need to release all dynamically allocated memory, for this we call the Node's clear function inside the Destructor to free memory assigned by us and make all pointers null.

3. Test Cases // TO DO

Test 1: Create a new tree by inputting a few locations and print it to check if they were inserted in the correct order or not, along with the size if it is correct too. Also, check if node with duplicate coordinates can be inserted or not.

Test 2: Try clearing all the contents of the tree and print it to check if it was properly cleared.

Test 3: After inserting a few nodes, try finding q_max, q_min and q_total to check if we get the correct result or not. Also search for node to check if it exists.

Test File Example 1:

```
i cityA;20.0;30.0;50000;300;400
i cityB;-10.0;35.0;60000;700;1000
i cityC;15.0;-30.0;7000;800;800
i cityD;-5.0;-7.0;100000;200;900
i cityE;-20.0;30.0;37000;100;1000
print
size
i cityF;-20.0;30.0;120000;950;600
```

Test File Example 2:

```
i cityA;20.0;30.0;50000;300;400
i cityB;-10.0;35.0;60000;700;1000
i cityC;15.0;-30.0;7000;800;800
i cityD;-5.0;-7.0;100000;200;900
i cityE;-20.0;30.0;37000;100;1000
print
clear
print
```

Test File Example 3:

```
i cityA;20.0;30.0;50000;300;400
i cityB;-10.0;35.0;60000;700;1000
i cityC;15.0;-30.0;7000;800;800
i cityD;-5.0;-7.0;100000;200;900
i cityE;-20.0;30.0;37000;100;1000
q_max 15.0;-30.0;NE;p
s -20.0;30.0
q_total 20.0;-40.0;SE;r
s 9.0;9.0
q_min -20.0;30.0;SW;s
```

4. Performance // TO DO

Insert function and search function has a time complexity of O(logn) for an average case when the tree might not be balanced.
Q_max, q_min and q_total functions follow Inorder traversal so they have a time complexity of O(n).
Clear will have a time complexity of O(n) because it has to traverse through the whole tree.
Size only returns the size of the tree so has time complexity of O(1).
Best time complexity of all functions would be O(1) when the tree is empty and there is no node in it. Then none of the functions would execute so the run time would be least.
Worst time complexity of all functions is O(N). This would be the case when the tree is unbalanced and exists only in one direction or in a similar pattern.