Project 2 Quad Tree

Bosco Han *y95han* 20651352

In this project, there are three cpp files

TreeNode.cpp: this file defines the node data structure of a quad tree. It stores all the necessary info and methods for a given node in our quad tree. It's got pointers to is children nodes and the data store in the current node. The class attributes and constructor are as follow:

```
class TreeNode {
    public:
    TreeNode *NE;
    TreeNode *NW;
    TreeNode *SW;
    TreeNode *SE;
    CityInfo info;

    TreeNode(CityInfo _info) :
        info(_info), NE(nullptr), NW(nullptr), SW(nullptr), SE(nullptr) {}

    TreeNode(CityInfo _info, TreeNode *_NE, TreeNode *_NW, TreeNode *_SW, TreeNode *_SE) :
        info(_info), NE(_NE), NW(_NW), SW(_SW), SE(_SE) {}
```

Also, in this class are methods related to manipulating the tree. They are as follows:

```
TreeNode* insertNode(TreeNode *root, CityInfo *info, bool foundDuplicates[]);
int countNodes(TreeNode *root);
void printInorder(TreeNode * root, vector<string> &list);
TreeNode* findInTree(TreeNode *root, double longitude, double latitude);
void findMax(TreeNode *root, double &max, string const& attribute);
void findMin(TreeNode *root, double &min, string const& attribute);
void findTotal(TreeNode *root, double &total, string const& attribute);
void clearAllNodes(TreeNode *root);
```

In **InsertNode**, the coordinates and attributes of the city are wrapped in the CityInfo object. The method then recursively calls itself until it finds either an existing city, or a null node. In which it will either not insert as the city being inserted already exists, or create a new node.

In **CountNode**, the root instance is passed in, the method recursively calls itself, while adding 1 to each addition onto the call stack. Thus returning the total node count in the tree.

PrintInorder, prints the tree by inorder traversal.

findInTree, recursively looks for a node stored in the tree. It is implemented very much like binary search tree traversal; however since we have a quad tree, the run time is $O(\log_4 n)$ instead of $O(\log_2 n)$ as found in binary search tree search where n = total # of nodes.

In **findMax, findMin,** and **findTotal,** we use inorder traversal to look thru the all the nodes of the tree, at each recursion stack, we update the result we are looking for (max, min, or total) according to the current node's value.

Finally, **clearAllNodes**, takes the root reference and recursively traverse and delete each node. We need to set the root reference to null and call 'delete root' as the nodes were allocated by calling 'new' thus we need to delete them from the heap.

Below is the CityInfo class and its constructor. A CityInfo object is created and attached to each node as the new nodes are created.

```
class CityInfo {

    public:
    string name;
    double longitude;
    double latitude;
    int population;
    double costOfLiving;
    double avgSalary;
    CityInfo(string _name, double _longitude, double _latitude, int _population, double _costOfLiving, double _avgSalary) {
        name = _name;
        longitude = _longitude;
        latitude = _latitude;
        population = _population;
        costOfLiving = _costOfLiving;
        avgSalary = _avgSalary;
    }
};
```

Lastly, the last class *qttest.cpp* takes care of all the parsing and outputs of the test files. It takes care of reading the inputs from testx.in and outputting them in the proper format. For example, taking care of the decimal/integer outputs when avgSalary and population are called respectively.

Overall, All of this quad tree's methods are implemented recursively. The runtime and space complexity varies between $O(\log_4 n)$ to $O(n)$.

$O(\log_4 n)$ ~ roughly the height of a complete quadtree for runtime and space complexity occurs when calling **InsertNode** and **findInTree**

$O(n)$ run time and space occurs for all other methods as we must traverse thru all the nodes in the tree.

For both, n = total # of nodes in tree