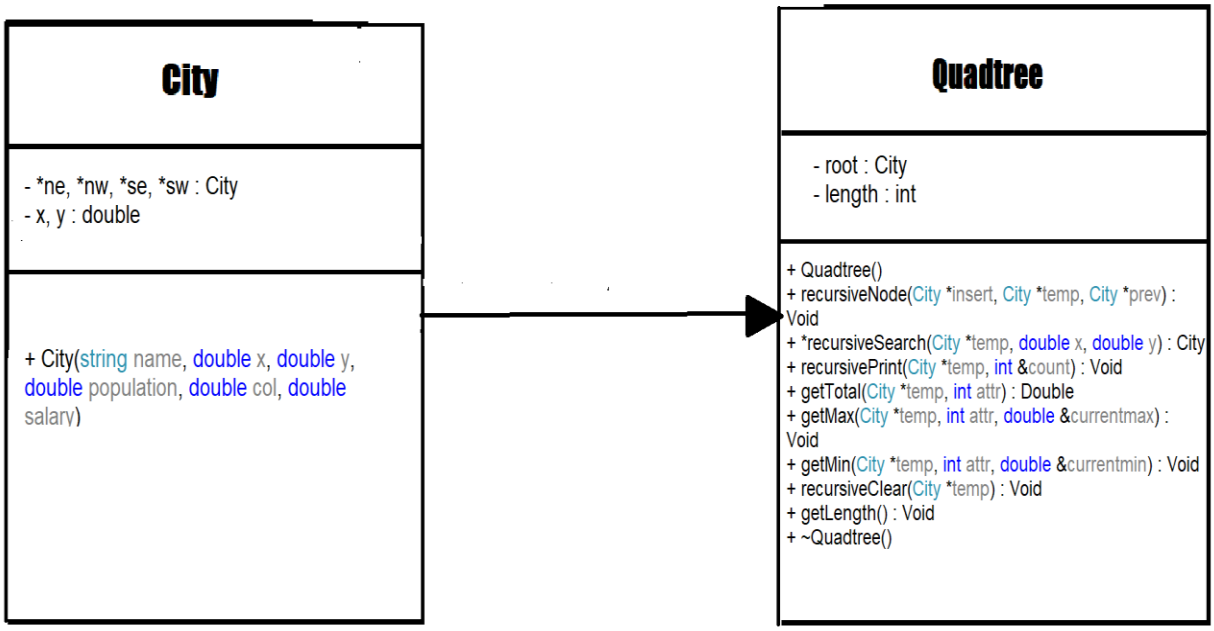
Overview of Classes

For this project, I used two classes for simplicity: “City.cpp” and “Quadtree.cpp”. Essentially, the City class is a node class. It stores the given properties of cities, as well as four pointers for NE, NW, SE, SW. The Quadtree class stores a root node, and defines all project functions such as insert, clear, p_max, etc. Once an instance of Quadtree is created in the “qttest.cpp” file, City nodes can be inserted. The first City inserted into the Quadtree will be made the root node, and all after will be linked to it. Explicit detail of each function can be seen below under “Function Design and Performance”.

UML Diagram



- UML diagram of Important functions in City and Quadtree classes

Function Design and Performance

City.cpp

For this class, I simply created class variables and a constructor.

Constructor

Once an instance of City is created, values for “string name”, “double x”, “double y”, “double population”, “double col”, and “double salary” are taken as constructor parameters. Also, upon instantiation the four pointers “City ne”, “City nw”, “City se” and “City sw” are set to be a null pointer, until they are later updated in the Quadtree class. All properties of City are public so they can easily be accessed and changed in Quadtree.

Quadtree.cpp

Most class functions for Quadtree have two parts, a caller and a recursive part. The caller function calls the recursive function and the recursive functions perform given operations (i.e. search). I did this so I could pass Quadtree's private root variable as a parameter to the recursive function without having to create a getter function.

Constructor

For Quadtree's constructor, no parameters are taken. For simplicity, upon insanitation two private class variables "City root" and "int length" are initialized. The "root" is variable is set to a null pointer and "length" is set to 0.

Insert (type Void) – O(log4n) time complexity:

For this function, I take in "City temp", "string name", "double x", "double y", "double population", "double col", and "double salary". The caller function first creates an instance of City, checks if root is null, and if not calls the recursive part of the insert process. For every iteration, if the current City node is not null, the function will call itself with NE, NW, SE, or SW as "temp", depending on the "x" and "y" coordinates. I made this function void because I am just changing an existing object (the current quad tree). The time complexity would be O(log4n) because at most the function needs to be called the for height of tree, h, times.

Search (type City) – O($\log_4 n$) time complexity:

Much like Insert, my Search function will traverse the tree until the proper node is found. This function takes parameters "City temp", "double x", and "double y". Once a node with matching coordinates is found in the recursive component, it is returned to the caller, with prints "found [name]". If a null pointer is reached (dead end), it is returned and "not found" is printed. Like Insert, this function compares the "x" and "y" with the current temp node each time so that temp->ne, temp->nw, temp->se, or temp->sw is called in the recursive case. This linearity allows for O($\log_4 n$) time.

P_Max, P_Min, P_Total (type Void, Void, Double) – O(n) time complexity

For these functions, I begin with a variable of type double in the caller function. This would be named "currentmax", "currentmin", "currenttotal", respectively. Also, before the recursion starts, the d value is checked, and NE, NW, SE and SW are entered accordingly. I then call the recursive function with parameter "City temp", "int attr", "double &currentmax/min/total". I converted "attr" to an int in "qttest.cpp" for simplify. The reference to the currentmax/min/total from the caller is changed on each iteration, thus changing the original variable. Because it is possible for the entire Quadtree to be checked, the worst case is O(n) time.

Destructor

To avoid memory leaks, I call the Clear function to delete everything in the tree before it goes out of scope. The Clear function deallocates all memory and sets null pointers.

Test Cases

- To begin testing, I simply tested the given test cases on the eceUbuntu server and had successfully results

- After this, these are some extra tested cases: deleting a city that wasn't there, clearing multiple times in a row, inserting a city will the same coordinates but different name, and printing an empty tree

- Next, I performed border case tests by setting populations/cost of living/salary to 0 or vary large numbers

- Lastly I tested for and corrected all memory leaks in my program