Design – P3

Kehan Xing
UW id: k5xing   Student#:20783902

Class:
I designed a class named Tree and a struct named City in this project.

Struct City:
The struct City contains six variables including name (string), x(double), y (double), p (int) which represents population, r (int) which represents cost of living and s (int) which represents salary.

Class Tree:
This class contains two variables and a struct named TreeNode to represent the nodes in the Quadtree. The two variables are root (TreeNode*) that represents the root of the tree and size (int). Inside the struct, there are five variables: city (City) which represents the city to be inserted in the tree, NW, NE, SW, SE (TreeNode*) which are used to connect the nodes. There is also a constructor, a destructor and nine functions in this class.

Member functions:
a. insert function: this is a function which can insert a leaf to the Quadtree, it requires a City parameter and a TreeNode* parameter which is used to implement recursive approach. The output of this function is a bool variable that represents whether the insertion is successful.

b. search function: this is a function which can check whether there is a city in the tree has the input position. It takes the position as two int parameters x (longitude) and y (latitude) and a TreeNode* parameter which is used to implement recursion. The function returns a TreeNode* variable which represents the node with the input position; if there's no such node, it will return NULL.

c. findnode function: this function can find the specific child of the input node, and it is used in the maximum, minimum, total function. It requires a TreeNode* parameter that represents the parent and a string variable which is the specific direction. The return type of this function is TreeNode* which is the child with the specific direction; if there's no child in that direction, it will return NULL.

d. maximum function: this function takes a TreeNode* variable as the root of the subtree and a string variable as the specific attribute to be compared, and it will find the maximum value in the subtree for that specific attribute. The return type of this function is int which is the maximum value.

e. minimum function: this function takes a TreeNode* variable as the root of the subtree and a string variable as the specific attribute to be compared, and it will find the minimum value in the subtree for that specific attribute. The return type of this function is int which is the minimum value.

f. total function: this function takes a TreeNode* variable as the root of the subtree and a string variable as the specific attribute to be summed, and it will find the total value in the subtree for that

specific attribute. The return type of this function is int which is the sum of the values.
g. print function: this function can print the Quadtree using an inorder traversal. It requires a TreeNode* parameter used for recursion, and there's no output (void).

h. clear function: this function can delete all the nodes in the. It requires a TreeNode* parameter used for recursion, and there's no output (void).

i. count function: this function is used to return the size of the tree. It takes no input, and it returns an int variable which is the size.

<u>Constructor and Destructor:</u>
There's one constructor and one destructor. The constructor is a default constructor which takes no input, and it set the variable size to 0 and the variable root to NULL. The destructor takes no input as well, and it deletes the tree by setting size to -1 and root to NULL.

<u>Design decision related to running time:</u>
In this project, it is expected to use recursive approach in all functions.
With the recursive approach, the running time of the commands in my program becomes:

a. i (insert): if balanced, $f(n) = f(n/4) + \Theta(1) \Rightarrow T(n) = \Theta(\log n)$
b. s (search): $f(n) = f(n/4) + \Theta(1) \Rightarrow T(n) = \Theta(\log n)$
c. q_max/q_min/total: $f(n) = 4f(n/4) + \Theta(1) \Rightarrow T(n) = \Theta(n)$
d. print: $f(n) = 4f(n/4) + \Theta(1) \Rightarrow T(n) = \Theta(n)$
e. clear: $f(n) = 4f(n/4) + \Theta(1) \Rightarrow T(n) = \Theta(n)$
f. size: $T(n) = \Theta(1)$

My program satisfies the expected time complexity in the project description.