ECE250 Project 3 Design Document- Trees

Devon Miller-Junk 20763552 November 8, 2019

Project 3: Trees

For the design of project 3, my plan was to create a quad tree that can perform search and insert operations in O(nlgn) if it is balanced. The clear, and traversal operations should take O(n) time.

**Note:** To compile and run my program please use the following

g++ -std=c++11 -o quadtreetest quadtreetest.cpp

./ quadtreetest <inputFile >outputFile

1. **The Quad Tree**

The Quad Tree is like a Binary Tree except that it has 4 children, identified by their coordinates instead of a single value. To select which quadrant a node should go into, the 2 sets of coordinates are compared to each other. Each node is a city in this implementation and contains information about the city such as its name, population, cost of living, and average net salary.

* 1. **Node Struct**

The Node struct contains all the information about each individual city and is combined with other nodes to build the quad tree.

* + 1. **Attribute enum**

The attribute enum is used by the traversal to select which data type to use. It can be used to focus on population, cost of living, or average net salary.

* + 1. **Type enum**

The type enum is used by the traversal to select which aggregate function to perform on the data. This can be selecting the min, selecting the max, or summing the values.

Each node contains a pointer to its parent (which is not needed for this implementation of a tree), and 4 pointers to each of it’s four children (quadrants). Each node also contains a location (it’s x,y coordinates) that is an array of 2 doubles. Lastly, each node has a name, population, average net salary, and cost of living associated with the city that the node represents. The name is a string, the other variables are long integers.

The Node struct has 2 constructors and one destructor. The simple constructor constructs a node with all pointers set to nullptr and all integers set to -1, and all strings set to and empty string. The other constructor takes values for all the properties of the node and fills them in (string for the name, double array for the location, and long integers for the population, cost of living, and average net salary), the parent and children pointers are all set to null pointer. The destructor of the node takes no parameters and deletes all the children of the node.

In the node struct, 2 operators were overloaded. The equals operator was overloaded and takes a pointer to a node as a parameter. It sets the properties of the current node equal to those of the node passed in. The equality operator was overloaded, with one function taking a pointer to a node as a parameter and the other function taking an array of doubles as a parameter. The functions return true if the nodes themselves are equal or the locations are equal.

The other functions in the node struct are insert, search, traversal, print, and select region. The insert function accepts a pointer to a node as a parameter and returns a true/false depending on if the node was inserted into the tree. The insert function recursively descends the tree until it reaches the bottom of the tree, where the node is inserted as a leaf. The insert function thus takes time depending on the height of the tree, which is Θ(lgn) for a balanced tree and Θ(n) in the worst-case scenario (the tree behaving like a linked list). The search function accepts an array of doubles representing a location as a parameter and recursively descends the tree until it reaches a node whose location is equivalent to the parameter, or it reaches a leaf. A pointer to the node is returned if the node is found, otherwise nullptr is returned. Since the search descends the tree, the maximum time it can take is proportional to the height of the tree (it would take less time if the node is found before it reaches the leaves), and would thus take Θ(lgn) for a balanced tree or Θ(n) for the worst case scenario for the height of the tree. The traversal function takes an attribute and a type as parameters and finds the min/max/total of the given attribute of the node and all its children. The result is returned as a long int. This is done recursively by calling traversal on each of the children of the node. This means that the traversal function takes Θ(n) time where n is the number of nodes that are in the sub-tree whose root is the specific node in question. The print function takes no parameters and returns a string. The print function traverses the tree using an in-order traversal and returns a string of the names of the cities in the tree separated by spaces (in the order described by the in-order traversal). Since the print function performs a constant time operation on each node of the tree, it takes Θ(n) time to complete. The select region simply compares the values of a given location (passed in as a double array) with the location of the current node. It returns an index (as an integer) representing the correct quadrant the location would fall under. This takes constant time.

* 1. **Quad Tree Class**

The Quad Tree class contains a pointer to the root Node and an integer size. The Quad Tree itself is just used to interface with the nodes to make them act like a tree.

The Quad Tree class has a simple constructor and destructor. The constructor takes no parameters and creates a quadtree with 0 size and sets the root pointer to nullptr. The destructor calls the destructor of the root node, which recursively deletes all nodes in the quad tree. This takes linear time, since it takes constant time to delete each individual node.

The Quad Tree class also contains 4 functions for interfacing with the nodes (insert, search, clear and print). The insert function takes a pointer to a node as a parameter and returns the Boolean returned by calling the insert function of the root node. It thus also takes time proportional to the height of the tree (Θ(lgn) if balanced, Θ(n) worst-case). The search function takes an array of doubles as a parameter and returns the pointer to a node returned by calling the search function on the root node. This function also takes time proportional to the height of the tree. The clear function calls the destructor on the root and the tree, sets the size of the tree to 0 and sets the root of the tree equal to nullptr. Since calling the destructor on the root node of the tree recursively deletes the entire tree, then the operation takes Θ(n) time (proportional to the number of elements in the tree). The print function takes no parameters and returns the string returned by calling the print function on the root node. Thus, it also takes linear time.

1. **Utility File**

I used a utility file to hold a few functions that are useful for string parsing and input handling. These functions just modify the strings received from the input file. The remove new line function takes a string and returns the same string with all new line characters removed. The get command function takes a string and returns the substring that exists before the first space in that string. The get inputs function takes a string and returns the substring that exists after the first space in that string. The separate by commas function takes a string and an integer size as parameters. It returns the input separated by commas into an array with length of the size parameter. The get direction function takes a string as a parameter and returns an integer index of the correct quadrant for that string (i.e. NE is 0).

1. **References**

Tahvildari, L. (2019, Fall). *ECE 250 Lectures*. *ECE 250 Lectures*. Waterloo.

Ward, P. (2018, Fall). *ECE 150 Lectures*. *ECE 150 Lectures*. Waterloo.