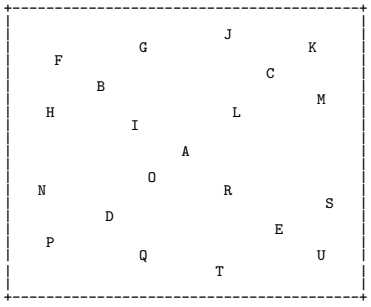
CSCI-1200 Data Structures — Fall 2017
Homework 8 — Quad Trees & Tree Iteration

In this assignment you will build a custom data structure named `QuadTree`. The data structure you will build for this homework is similar to the classic quad tree, octree, k-d tree, and binary space partition data structures from computational geometry. These structures are used to improve the performance of applications that use large spatial data sets including: ray tracing in computer graphics, collision detection for simulation and gaming, motion planning for robotics, nearest neighbor calculation, image processing, and many, many others. Our `QuadTree` implementation will share some of the framework of the `ds_set` implementation we have seen in lecture and lab. You are encouraged to carefully study that implementation as you work on this homework.

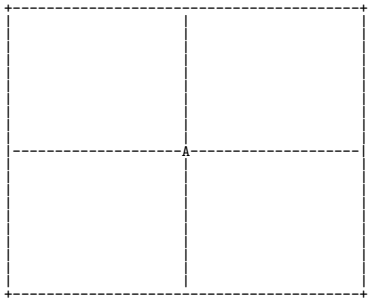
The diagrams below illustrate the incremental construction of the `QuadTree` data structure. In this example, we add the 21 *two-dimensional points* shown in the first image to the tree structure. We will add them in the alphabetical order of their *labels*. Each time a point is added we locate the rectangular region containing that point and subdivide that region into 4 smaller rectangles using the *x,y* coordinates of that point as the vertical and horizontal dividing lines.

Each 2D coordinate (x, y) is stored in the `Point` class. In these plots (0,0) is in the upper left corner. The x axis runs horizontally, with increasing values to the right. The y axis runs vertically with increasing values in the downward direction.

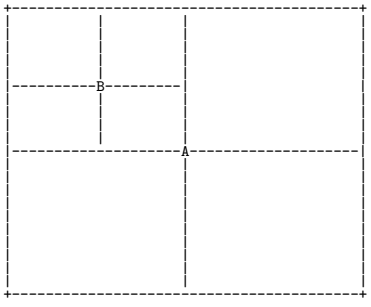
input points



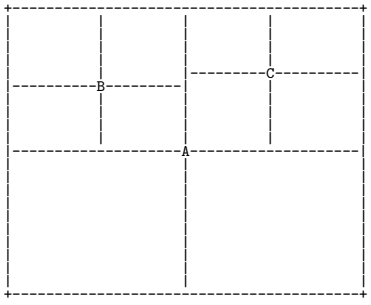
after adding the 1^{st} point



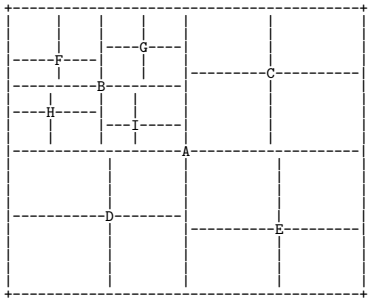
after adding the 2^{nd} point



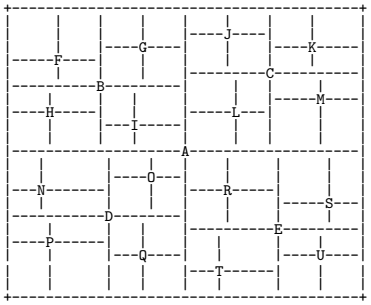
after adding the 3^{rd} point



after adding 9 points



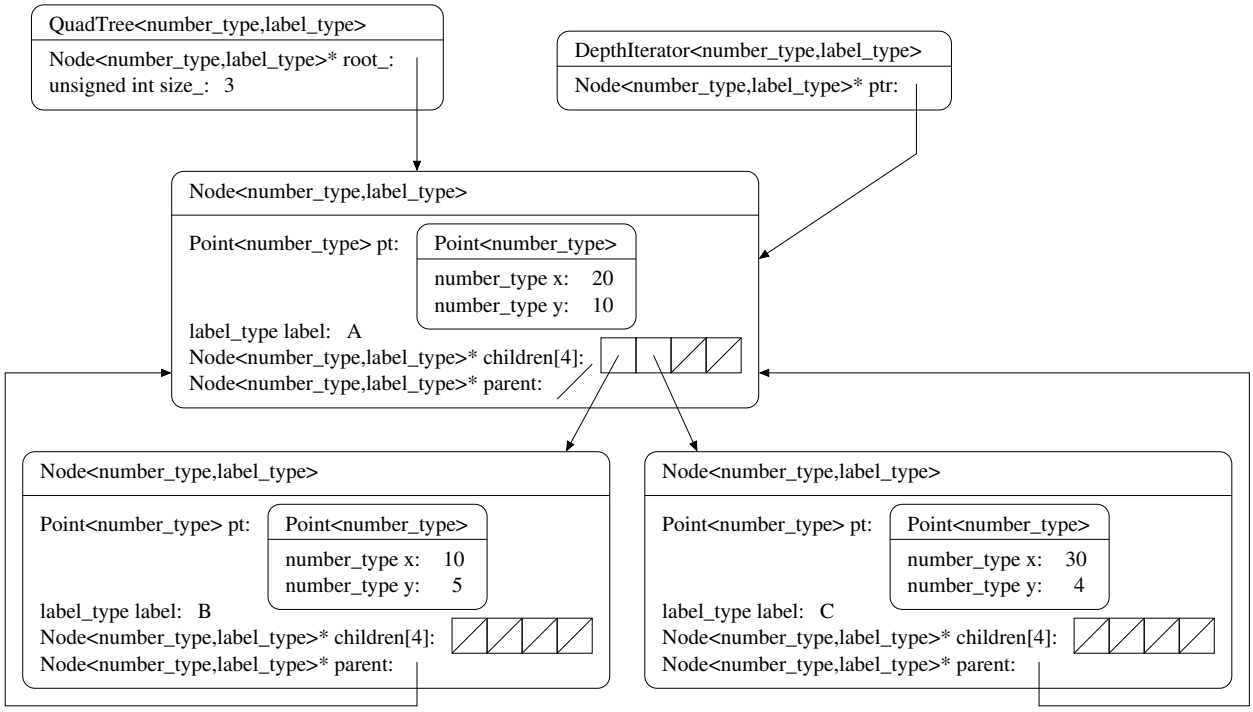
after adding all 21 points



Like an STL `map` and STL `set`, inserting a new `Point` into the `QuadTree` or querying (using `find`) whether a `Point` is already in the structure is fast, and can be completed in O(log n), where n is the number of `Points` in the tree. However, unlike the `set` and `map` structures which are based on a *binary* tree, having two subtrees per node, the `QuadTree` has 4 children at each node!

The data in our `QuadTree` can be visualized using two different output routines, `plot` and `print_sideways`. You will also implement two different methods for iterating over the tree: *depth-first* (specifically *pre-order*) or *breadth-first*. To do this, you will write two helper classes `DepthIterator` and `BreadthIterator` which will be `typedef`-ed within the `QuadTree` class.

Here is a diagram showing the relationships between the different classes you will implement for this assignment. Be sure to draw plenty of your own diagrams as you work, and be prepared to show these diagrams when you come to office hours or ALAC tutoring to ask for help on the assignment.



Note that the `QuadTree` is templated over two types, the *number_type* and the *label_type*. In this example, the number type is `int` and the label type is `char`. This is the detailed internal memory representation after adding the first 3 points in the example on the previous page. We also show a sample `DepthIterator` object initially attached to the root node of this tree. You should follow this diagram exactly, using these specific class and member variable types and names and placeholder template type names. We do not present the details of the `BreadthIterator` class in this diagram. You may design the internal representation of that class to complete the specified functionality.

Implementation

Your task for this homework is to implement the structure diagrammed above. We recommend that you begin your implementation by following the structure of the `ds_set` class we studied in lecture and lab. You will need to make a number of significant changes to the code, but the overall design: a "manager" class (`QuadTree`), the `Node` class, and `tree_iterator` classes is similar.

We provide the `Point` class and the implementation of the two `QuadTree` member functions for printing: `plot` and `print_sideways`. You will build the rest of your implementation around this starter code. Remember that because this is a *templated* class, you will not separate the implementation into `.h` and `.cpp` files. Keep all of your implementation in a single file named `quad_tree.h`, but make sure it is well-organized and appropriately commented.

The provided code in `main.cpp` illustrates the basic functionality of the `QuadTree` class including the `QuadTree` functions: `size`, `insert`, `find`, `height`, `begin`, `end`, `bf_begin`, and `bf_end`, and iterator functions: `operator++` (both pre- and post- increment), `operator*` (dereference), `getLabel`, and `getDepth`. Study these examples carefully to deduce the expected argument and return types of the functions. As this is a class with dynamically-allocated memory, you will also need to implement, test, and debug the copy constructor, assignment operator, and destructor. The homework server will compile and run your `quad_tree.h`

file with the instructor's solution to test your implementation of these functions. It will test your code with Dr. Memory and your program must be memory error and memory leak free for full credit.

We encourage you to work through the test cases in the provided `main.cpp` step-by-step, uncommenting and debugging one test at a time. The provided test cases do not adequately test all corner cases, so as you work, add your own test cases to the `student_tests` function. Be sure to test your templated implementation with other number types (e.g., `float`, `double`) and other label/data types (e.g., STL `string`, int, etc.). Note that the ASCII art `plot` function is only designed to work with small non-negative integers and `char` labels. Furthermore, the `plot` function assumes that no two points have the same x coordinate or the same y coordinate.

Extra Credit: Tree Balancing

How does the point insertion order affect the shape of the resulting `QuadTree` object? What are specific examples of the worst case and best case? Discuss in your README.txt file. For extra credit, you can implement a function named `BalanceTree` to re-order a point collection before inserting the data into the tree to improve the quality of the resulting tree. `QuadTree` quality can be defined with various metrics including: minimal tree height, more equal partitioning of data into the 4 subtrees (approximately the same number of elements in each child tree), and rectangular subregions that have approximately equal area or that have more equal ratio of height to width. Certainly for some input collections you cannot simultaneously satisfy all of these properties! You may define how to prioritize these metrics. Be sure to write challenging test cases that show off your implementation.

Performance

Assuming our tree holds n points that are well-distributed and have been inserted into the structure in a random order to produce a generally well balanced tree, what is the order notation for running time and memory usage of the different operations of the `QuadTree`? Put your answers and a short justification for each answer in your `README.txt` file.

Homework Submission

Use good coding style and detailed comments when you design and implement your program. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.