ECE250 Project 3: Quadtrees

Due Date: Monday, November 20th at 11pm to the dropbox on learn

Overview

A quadtree is a type of "space partitioning tree" that enables fast lookup of points in a space. A quadtree is optimized specifically for 2D space. Each node represents a rectangular region in space bounded by the points (x_o, y_o) and (x_1, y_1). Within each node is an array of points of known, fixed size, given by (x, y) coordinates. Typically, these points represent important locations within the 2D space under study, with the rest of the space considered to be irrelevant. The rule is that if a node contains a specific point, then its coordinates fall within the rectangular region bounded by the node. Once the node's array is full, the tree grows as follows:

1. Add exactly four children to the node
2. The bounding rectangles for each child comprise exactly ¼ of the original bounding rectangle. That is, the range on both x and y are split exactly in half, giving us four identically sized sub-regions in space
3. The points stored in the parent node are distributed amongst the children according to their location in space
4. The parent node's point array is empty and will no longer be used to store points

This means that only the leaf nodes ever contain points, with internal nodes being used to store space partitioning data that can be used to efficiently search for points.

In this project you will be creating an implementation of a quadtree, and using it to perform a range-based search (see below). You must write the required classes yourself. Other than std::string, **you are not permitted to use any STL libraries.**

Understanding Range-Based Search

In a range-based search, we are interested obtaining a list of points within a given range. For example, I might want to know all points whose coordinates lie within the range $x \in (0,1), y \in (-1, -0.4)$. If we are guaranteed that all search regions are contained entirely within a single node's rectangular region of space, this problem is simple. However, if the search region intersects multiple nodes' rectangles, the problem is more difficult. Recursion or stack-based algorithms are often used for this purpose. You are expected to investigate options for algorithms to solve this problem, and implement one.

Program Design and Documentation

You must use proper Object-Oriented (OO) design principles to implement your solution. You will create a design using classes which have appropriately private/protected data members and appropriately public services (member functions). It is not acceptable to simply make all data members public. **Structs are not permitted.**

Write a short description of your design to be submitted along with your C++ solution files for marking according to the template posted to Learn.

Input/Output Requirements

Write a driver program that contains your main function. The driver program must read commands from standard input and write the output to standard output. The commands are described below. **All input files end with the string "EXIT", which should end the program and produce no output.**

Command	Parameters	Description	Output
INIT	m x0 y0 x1 y1	Create a new quadtree. The parameter m, which is guaranteed to be greater than 4, represents the number of points that can be stored at a node before the quadtree expands. The points (x0,y0), (x1,y1) represent a rectangle	**success** if x0 < x1 and y0 < y1 **illegal argument** (see below this table) if the success criteria are not met

		in space. This initial rectangle defines all of space for the quadtree – no point outside of this space may be stored. The boundary of this rectangle is included in the space of points. This command is guaranteed to be called exactly once at the beginning of each test case.	
INSERT	x y	Insert a point into the quadtree with co-ordinates (x,y), resize if needed. In case a point falls on the vertical boundary between the rectangles defined by two nodes, the point should be placed in the leftmost node. When a point falls on the horizontal boundary defined by two nodes, the point should be placed in the topmost node. If a point is at the exact intersection point of four nodes, the left, top node is to be used.	**success** if the point is within the range of quadtree **failure** if the point is not within the range of the quadtree or the exact point already exists
SEARCH	x y d	Determine if there is at least one point stored in the quadtree with distance strictly less than d from the point (x,y)	**point exists** if at least one such point exists **no point exists** if there is no such point
NEAREST	x y	Determine the nearest point stored in the quadtree from the point (x,y). Note: a solution to the "RANGE" input may help to solve this problem. In the case that there are multiple nearest points (for example, if the quadtree has points (0,0) and (1,1) and NEAREST 0.5 0.5 is called, choose the point whose x value is larger. In the case of a tie on the x values, choose the point whose x and y values are both larger.	**xp yp** the coordinates of the nearest point in the quadtree to the given point **no point exists** if the quadtree is empty
RANGE	xr0 yr0 xr1 yr1	Print a list of points within the given range. The points (xr0,yr0), (xr1,yr1) represent a rectangle in space and is subject to the same restrictions as in the INSERT command above. The boundary of the rectangle is not included in the range (that is, you should be using a strict less than relationship)	**xp0 yp0 xp1 yp1…** If the range is valid and points exist there, a list of the coordinates of the points. The order does not matter as long as, for each point, x is printed before y. Use spaces to separate points and a newline at the end of this line. **no points within range** if the range is valid but no points exist there **illegal argument** (see below this table) if the range is not valid
NUM		Output the total number of points stored in this quadtree	N The number of points stored in the quadtree
EXIT		All input files end with this string	This command produces no output

There are several ways to solve this problem that are valid.

You must analyze the runtime of your algorithms in your design document. Prove that your INSERT functionality can operate in O(D) time, where D is the depth of the quadtree. Prove that the RANGE function has runtime O(D) time whenever the search range falls entirely within the rectangle described by a single leaf node.

Illegal arguments: If necessary, your code must throw an illegal_argument exception, catch it, and output **"illegal argument"** (without quotes and followed by a newline) if it is caught, then continue processing subsequent lines of input. You may assume that the input format is correct. To do this, you will need to:

a. Define a class for this exception, call it **illegal_exception**
b. Throw an instance of this class when the condition is encountered using this line:

```
throw illegal_exception();
```

c. Use a try/catch block to handle this exception and print the desired output of the command

Test Files

Learn contains some examples input files with the corresponding output. The files are named test01.in, test02.in and so on with their corresponding output files named test01.out etc.

Valgrind and Code structure

15% of the grade of this project will be allocated to memory leaks, memory errors, and code structure. We will be using the Valgrind utility to do the memory check. The expected behaviour of Valgrind is to indicate 0 errors and 0 leaks possible, with all allocated bytes freed. To test your code with Valgrind, presuming you are using an input file test01.in, you would use the following command:

```
valgrind ./a.out < test01.in
```

Your code will also be subject to a static analysis to determine how well it is formatted, organized, and written.

Submitting your Program

Once you have completed your solution and tested it comprehensively on your own computer or the lab computers, you should transfer your files to the eceUbuntu server and test there. We perform automated testing on this platform, so if your code works on your own computer but not on eceUbuntu it will be considered incorrect. **A makefile is required for this project since the exact source structure you use will be unique to you.**

Once you are done your testing you must create a compressed file in the tar.gz format, that contains:

- A typed document, **maximum of three pages**, describing your design. Submit this document in PDF format. The name of this file should be xxxxxxxx_design_pn.pdf where xxxxxxxx is your maximum 8-character UW user ID and n is the project number (e.g., mstachow_design_p3.pdf).
- Your test program, containing your main function.
- Required header files that you created.
- Any additional support *.cpp files that you created.
- A makefile, named Makefile, with commands to compile your solution. Your executable must be named a.out.

The name of your compressed file should be **xxxxxxxx_p3.tar.gz**, where xxxxxxxx is your UW ID as above.