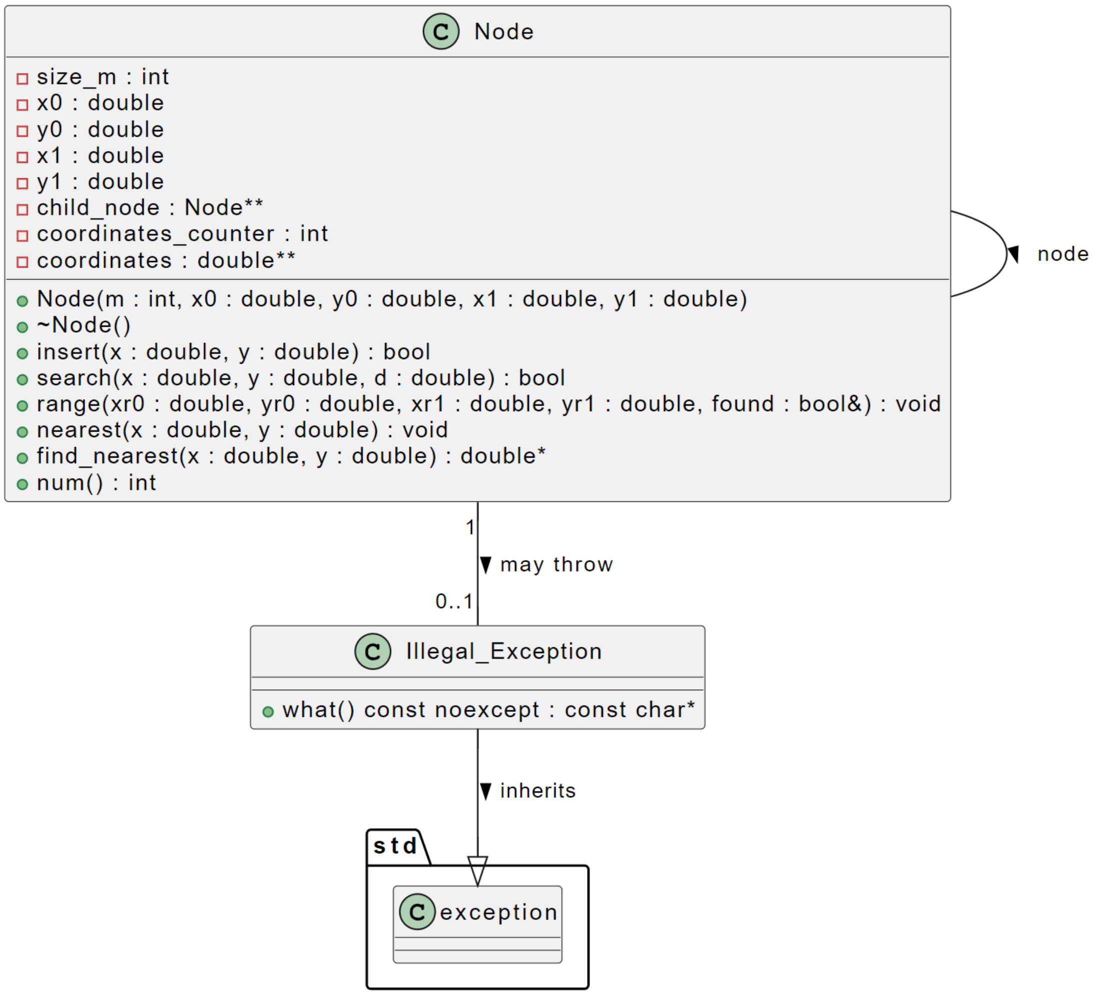
ECE250 LAB3 Design Document

Author: Bowen Zheng, student #: 20949303, student ID: b57zheng

Class UMLs


Node
size_m : int
x0 : double
y0 : double
x1 : double
y1 : double
child_node : Node**
coordinates_counter : int
coordinates : double**
Node(m : int, x0 : double, y0 : double, x1 : double, y1 : double)
~Node()
insert(x : double, y : double) : bool
search(x : double, y : double, d : double) : bool
range(xr0 : double, yr0 : double, xr1 : double, yr1 : double, found : bool&) : void
nearest(x : double, y : double) : void
find_nearest(x : double, y : double) : double*
num() : int
node
1
may throw
0..1
Illegal_Exception
what() const noexcept : const char*
inherits
std
exception

Node Class:

- <u>Constructors:</u>

    The overloaded constructor was designed to support the creation of nodes. it throws exception if the quadtree boundary is invalid and ensures all objects start in a consistent state and the allocated memory addresses don't store garbage.

- <u>Destructor:</u>

    The destructor was designed to deallocate memories allocated to the quadtree at the end of the program execution, which prevents memory leaks.

- <u>Class Design:</u>

    The Node class in the project is a key component of a quadtree. A significant design choice is the use of dynamic memory allocation for both the points and the child nodes (child_node). This approach allows for efficient memory usage, as child nodes are only created, when necessary (i.e., when the node exceeds its capacity), demonstrating a lazy initialization strategy.

    Core functionalities, such as inserting points, searching, and nearest-neighbor queries, are encapsulated within the class. The class also handles boundary checks and illegal arguments through exceptions. Overall, this Node class design is focused on spatial efficiency and flexibility.

- <u>Functions: Insert</u>

The decision to implement lazy subdivision in the insert function is a significant design choice. The node only subdivides when it exceeds its capacity (size_m), preventing unnecessary memory usage and computational overhead. The function then recursively inserts existing points into the appropriate child nodes based on their spatial location.

  - Run Time Analysis:
    - At each level of the tree, the insert function performs a constant amount of work. This includes checking whether the point lies within the node's boundaries and, if necessary, determining the correct child node for insertion. Both of these operations are O(1).
    - In the worst case, inserting a point requires traversing from the root to a leaf node. The depth of this traversal D is the maximum number of recursive calls made by the insert function.
    - Since each level of recursion involves O(1) work and there are at most D levels to traverse, the total work done by the insert function is the sum of O(1) operations across D levels. Hence, the upper limit of the runtime is O(D).

- <u>Function: search</u>

The search function uses a distance-based algorithm. When searching for a point within a certain distance, the function first checks if the search area intersects with the current node. If there's an intersection, and if the node has children, the function recursively searches within these child nodes. This recursive descent allows the function to quickly narrow down the search area, avoiding unnecessary checks in regions of the quadtree that do not intersect with the search radius.

- <u>Function: range</u>

The range function employs recursion to handle range queries. When given a rectangular range, the function checks for intersection with the node's boundaries. If an intersection is detected and the node has children, the function is called recursively on each child node. This process efficiently prunes the search space, as branches of the quadtree that do not intersect with the range are not explored, enhancing query performance.

  - Run time Analysis:
    - When the search range is entirely within the bounds of a single leaf node, the range function needs to traverse from the root to this specific leaf node. The depth of this traversal, O(D), represents the maximum number of recursive calls made by the function.
    - At each level of the tree, the range function performs an O(1) operation to determine whether to continue the traversal towards a child node. This involves checking if the search range intersects with the bounds of the current node. If there's no intersection, the traversal stops at that level; otherwise, it continues to the appropriate child node. Since the search range is entirely within a leaf node, only one path from the root to the leaf is explored.
    - Upon reaching the leaf node that fully contains the search range, the function may perform a linear scan of the points within this node. This is an O(1) operation, given that each node has a fixed capacity M.
    - Since each level of the tree incurs O(1) work and there are at most D levels to traverse, the upper limit of the run time of the range function is O(D).

- <u>Function: find_nearest</u>

The design decision for implementing find_nearest function as a helper function to nearest function is to avoid printing errors in the recursive algorithm. The algorithm for the find_nearest function is designed to exploit the hierarchical structure of the quadtree. It recursively navigates through the nodes, comparing distances to find the nearest point. This approach leverages the spatial partitioning of the quadtree to limit the search area for better efficiency.

- Function: nearest

  This wrapper function integrates the find_nearest functionality and handles the display of error messages to the console, ensuring a seamless operation.

- Function: num

  This function using recursion to count the points in a node and its children, and output the total number of points store in the quadtree.

Illegal_Exception Class:

- Class Design:

  This class was designed as a custom exception for the quadtree implementation, inherits from the standard std::exception in C++. This design choice ensures compatibility with C++'s exception handling framework while providing more specific error messages related to quadtree operations. By overriding the what() method, it delivers pre-specified error information.

- Function: what:

  The what() function in the Illegal_Exception class, overriding the same method from the base std::exception, is tailored to deliver pre-specified error messages pertinent to the quadtree context. This design choice ensures compatibility with standard C++ exception handling while providing clear, context-relevant error information.