Implement Polygon range-searching using kd-trees

GROUP MEMBERS:

1) JANVI POLAM REDDY,
2) BHAVYA HALASABELE SHIVAKUMARAIAH

The project implements a query algorithm that returns count and the indices of the points inside the query polygon. Our project structure includes main.cpp file where the entire code logic has been implemented. We made few changes in the ANN library file's header to include Point and Polygon classes. We have written our code in such a way that it checks for all the folders starting with test within the directory where our project is present and generates 2 output files, one for the count and the other for reporting the indices. These files are compared with the given files in the testcase folder and the accuracy is printed out. We used the ray casting algorithm to check if the point is inside the polygon or not.

Details of main.cpp functions:

1. void writeInsideCountsToJson(const std::vector<unsigned int>& insideCounts, const std::string& outputFilePath)
   - This function takes a vector of unsigned integers (insideCounts) and a file path (outputFilePath) as input.
   - It converts the insideCounts vector to a JSON array using the nlohmann::json library.
   - It opens the output file specified by outputFilePath using std::ofstream.
   - If the file is successfully opened, it writes the JSON array to the file.
   - If the file cannot be opened, it prints an error message to the console.

2. void writeInsidePointsIndicesToJson(const std::vector<std::vector<unsigned int>>& insidePointsIndices, const std::string& outputFilePath)
   - This function takes a vector of vectors of unsigned integers (insidePointsIndices) and a file path (outputFilePath) as input.
   - It converts the insidePointsIndices vector of vectors to a JSON array using the nlohmann::json library.
   - It opens the output file specified by outputFilePath using std::ofstream.
   - If the file is successfully opened, it writes the JSON array to the file.
   - If the file cannot be opened, it prints an error message to the console.

3. void loadFromFile(const std::string& pointFilePath, const std::string& queryFilePath, std::vector<Point>& points, std::vector<std::vector<Point>>& polygons)
   - This function takes two file paths (pointFilePath and queryFilePath) and two reference parameters (points and polygons) as input.

- It opens the pointFilePath file using std::ifstream, parses the JSON data using json::parse(), and reads the points from the parsed data.
- For each point in the JSON data, it creates a Point object with an index and coordinates and adds it to the points vector.
- It closes the pointFilePath file.
- It opens the queryFilePath file using std::ifstream and parses the JSON data using json::parse().
- It reads the polygon vertices from the parsed data and creates polygons.
- For each polygon in the JSON data, it creates a vector of Point objects representing the vertices and adds it to the polygons vector.
- It closes the queryFilePath file.

4. bool containsPoint(const std::vector<Point>& vertices, const Point& p)
   - This function takes a vector of Point objects (vertices) representing the vertices of a polygon and a Point object (p) as input.
   - It determines whether the point p is inside the polygon defined by the vertices using the ray casting algorithm.
   - It initializes a count variable to keep track of the number of intersections.
   - It iterates over each edge of the polygon (defined by consecutive vertices) and performs the following checks:
   - If the point is to the left of the edge when moving from vertex i to vertex j, it calculates the x-coordinate of the intersection and increments the count if the point's x-coordinate is less than the intersection's x-coordinate.
   - If the point lies on the line segment defined by vertices i and j, it returns true indicating that the point is inside the polygon.
   - Finally, it returns true if the count is odd (indicating an odd number of intersections) and false otherwise.

5. void testRangeSearchFromFile(const std::string& pointFilePath, const std::string& queryFilePath, const std::string& testFolderPath)
   - This function takes file paths (pointFilePath and queryFilePath) as input.
   - It starts a timer for preprocessing.
   - It declares vectors to store the points and polygons.
   - It calls the loadFromFile function to load the points and polygons from the specified files.
   - It converts the points to the ANNpointArray format required by the ANN library.
   - It builds a kd-tree using the converted data points.
   - It stops the preprocessing timer and calculates the preprocessing time.
   - It starts a timer for the query.
   - It performs the range search queries for each polygon:
   - It creates a Polygon object from the polygon vertices.
   - It defines the bounding box for the range search based on the polygon's dimensions.
   - It performs the range search within the bounding box using a large radius.
   - It checks which of the points found in the range search are actually inside the polygon using the containsPoint function.

- It stores the count of inside points and their indices in separate vectors.
- It cleans up the allocated memory.
- It stops the query timer and calculates the query time.
- It writes the inside counts and inside points indices to JSON files using the writeInsideCountsToJson and writeInsidePointsIndicesToJson functions.
- It compares the generated output files with the given output files.
- It prints the accuracy results, preprocessing time, query time, and total time.

6. Int main()
    - The main function is the entry point of the program.
    - It Iterate over entries in the base path and get the folders starting with test and then calls the testRangeSearchFromFile function with the file paths for the input points, query polygons and testFolder path.
    - It returns 0 to indicate successful program execution.

Details of Point.cpp file:

Point class is used for representing points in 2D space.
Member Variables:
id: An integer ID for the point (potentially for unique identification).
annPoint: An ANNpointArray allocated with size 2, assuming points are represented in 2D space using x and y coordinates.
Constructors:
Point(int id, ANNcoord x, ANNcoord y): This is the constructor that initializes a new Point object with a given ID, x-coordinate, and y-coordinate. It also allocates memory for the annPoint array using annAllocPt.
Point(const Point& point): This is a copy constructor that performs a deep copy of another Point object. It copies the ID and allocates new memory for the annPoint array, then copies the x and y coordinates from the original point.
Destructor:
~Point(): The destructor deallocates the memory associated with the annPoint array using annDeallocPt to avoid memory leaks.
Static Function:
ConvertToANNpointArray(const std::vector<Point>& points, int n): This static function takes a vector of Point objects and their size (n) as input. It allocates memory for an ANNpointArray (pa) of size n and iterates through the points vector, copying the x and y coordinates from each Point object into the corresponding element of the pa array. The function then returns the created ANNpointArray.

Details of Polygon.cpp file:

Polygon class represents a polygon in 2D space.
Member Variables:
m: An integer representing the number of vertices in the polygon.
vertices: A std::vector<Point> to store the collection of Point objects representing the polygon's vertices.

low: An ANNpointArray allocated with size 2, storing the minimum x and y coordinates of the polygon (bounding box).
high: An ANNpointArray allocated with size 2, storing the maximum x and y coordinates of the polygon (bounding box).
Constructor:
Polygon(std::vector<Point>& polygon_vertices, int m): This constructor initializes a new Polygon object. It takes a reference to a std::vector<Point>containing the polygon's vertices and their size (m) as input.
It performs a deep copy of the vertices:
Clears the vertices vector in the Polygon object to avoid duplicates.
Iterates through the input polygon_vertices and creates new Pointobjects within the Polygon object's vertices vector using the copy constructor.
It sets up the bounding box:
Allocates memory for low and high using annAllocPt assuming a 2D space.
Initializes low and high with the coordinates of the first vertex in the polygon.
Iterates through the remaining vertices and updates low and high values to capture the minimum and maximum x and y coordinates of all vertices, defining the bounding box of the polygon.
Destructor:
~Polygon(): The destructor doesn't explicitly deallocate the memory for the vertices vector. This is because the assumption is that the memory for the Point objects within the vertices vector is managed elsewhere (likely in the loadFromFile function). However, it does deallocate the memory associated with the low and high bounding box arrays using annDeallocPt to avoid memory leaks.

Compiling and Execution:

To compile the project, download the zip file, extract it and then type make once you are inside the project folder. After executing the make command run the rangesearch executable which is generated using the command “./rangesearch”. This will execute the code for all the testcases and prints the output.

Extra Credit Option:

Create Doxyfile inside project directory
doxygen -g Doxyfile

Run command to create html doc
doxygen Doxyfile

Run command to open html file in browser. Before, go to the directory which has html file.
/usr/bin/python3 -m http.server