COMP6210: Report for Assignment 2

Coordinator_ID_Surname

1. Group Member Information

Name	ID	Email:	Assigned Task
Coordinator: Muntasir Md Nafis	48312932	muntasirmd.nafis@students.mq.edu.au	1
Mohammad Marzan Rahman	45853789	Mohammadmarzan.rahman@students.mq.edu.au	2
Ridwan Bin Khalid	48457515	Ridwanbin.khalid@students.mq.edu.au	3

2. Program Execution Requirements

2.1 Program Environment (e.g., OS, database, CPU, etc.)

OS: macOS (M1)
Hardware:

- **CPU**: Apple M1 (ARM architecture)
- **Memory**: 8GB Unified Memory
- **Storage**: 256GB SSD
  Python version: 3.10.13
  Packages:
- Math (included with Python)
- OS (included with Python)
- Time (included with Python)
- Create_rtree (uploaded with code)

2.2 Input Files and Parameters (directory settings and other parameters)

Task 1:

Filename: Task1.py

Parameters: N/A – No parameters required when running the python file.

Input files: Parking_dataset.txt

Task 2:

Filename: Task2.py

Parameters: N/A – No parameters required when running the python file.

Input Files: city2.txt

2.3 Additional Requirements

Ensure datasets are within the same directory as the main program files and current directory has been set so the datasets are visible to the programs.

3. Program Documentation

3.1 Program Organization – Part 1 (Task 1)

If your assignment consists of multiple files and/or classes, please provide brief, high-level descriptions of each file/class within your program, as illustrated below.

Class/File Name	**Description (detailed information)**
task1.py	This is the main script responsible for implementing nearest neighbor search algorithms: Sequential Scan, Best First Search using R-tree, and Divide-and-Conquer using two R-trees. It handles reading input data, performing the searches, and writing results to an output file.
create_rtree.py	Contains the implementation of the R-tree structure, including the RTree and Node classes. This file handles insertion, MBR (Minimum Bounding Rectangle) management, node splitting, and overflow handling.

3.2 Function Description – Part 1 (Task 1)

Task1.py

Function Name **(parameters)**	**Description (detailed information)**
euclidean_distance(p1, p2)	Calculates the Euclidean distance between two points p1 and p2 using the formula: $\sqrt{((x2 - x1)^2 + (y2 - y1)^2)}$. This is used to measure distances in nearest neighbor searches.
sequential_scan(query, data)	Implements the Sequential Scan method to find the nearest neighbor by iterating through all data points and comparing distances to the query point. Returns the nearest point.
best_first_search(query, rtree)	Performs the Best First Search algorithm using the R-tree structure to find the nearest neighbor. Uses a stack to traverse the tree, comparing distances between the query point and data points.
divide_and_conquer_search (left_rtree, right_rtree, query)	Utilizes two R-trees to execute a Divide-and-Conquer approach for finding the nearest neighbor. Compares results from both R-trees and returns the closest point.
load_dataset(filename)	Reads a dataset from a specified text file and returns a list of data points as dictionaries with id, x, and y coordinates.
main()	Orchestrates the program flow by loading datasets, building R-trees, executing the search algorithms (Sequential Scan, Best First Search, and Divide-and-Conquer), measuring execution times, and writing results to an output file.

create_rtree.py

Function Name (parameters)	Description (detailed information)
main(points_list)	Constructs an R-tree using the provided list of points and returns the tree. Handles the process of inserting points into the tree while maintaining the MBR and splitting nodes when needed.
Node.__init__()	Initializes a new Node instance with attributes for id, child_nodes, data_points, parent, and MBR. Prepares the node to function as a leaf or internal node.
Node.perimeter()	Calculates the half-perimeter of the node's MBR. Used during the insertion process to determine which subtree to expand.
Node.is_overflow()	Checks if the number of data points or child nodes in a node exceeds the maximum capacity (B). Returns True if the node is overflowing.
Node.is_root()	Returns True if the node is the root of the R-tree (i.e., has no parent).
Node.is_leaf()	Returns True if the node is a leaf (i.e., has no child nodes).

RTree.__init__()	Initializes an R-tree by creating an empty root node.
RTree.insert(u, p)	Inserts a data point p into the R-tree starting from node u. Updates MBRs and handles node overflow as necessary, ensuring efficient space management.
RTree.choose_subtree(u, p)	Selects the best subtree for insertion by minimizing the increase in perimeter of the MBR. Returns the chosen child node for further insertion.
RTree.peri_increase(node, p)	Calculates the increase in perimeter of the MBR when adding a new data point to a node. Helps determine the best insertion path to minimize expansion.
RTree.handle_overflow(u)	Splits an overflowing node into two new nodes and handles restructuring. Creates a new root if necessary. Ensures the tree maintains balance.
RTree.split(u)	Splits an overflowing node u into two new nodes. Selects the optimal split by evaluating the smallest perimeter sum. Updates parent references for the new nodes.
RTree.add_child(node, child)	Adds a child node to a parent node and updates the parent’s MBR accordingly.

RTree.add_data_point(node, data_point)	Adds a new data point to a leaf node and updates the node’s MBR.
RTree.update_mbr(node)	Updates the MBR of a node to accurately cover all its data points or child nodes. Ensures proper spatial representation of the node’s content.

4. Program Documentation

4.1 Program Organization – Part 2 (Task 2)

Class/File Name	Description (detailed information)
task2.py	Main script for executing various skyline search algorithms: Sequential Scan, BBS algorithm using R-tree, and Divide-and-Conquer approach. Handles reading input data, executing the searches, and writing the results to an output file.
create_rtree.py	Implements the R-tree structure, including the RTree and Node classes. Manages insertion, MBR (Minimum Bounding Rectangle) updates, node splitting, and overflow handling. Used for spatial indexing in skyline search algorithms.

4.2 Function Description – Part 2 (Task 2)

Task2.py

Function Name (parameters)	**Description (detailed information)**
read_dataset(file_path)	Reads a dataset from a file and returns a list of points as dictionaries with id, x, and y coordinates. Used to load input data for skyline algorithms.
dominates(a, b)	Checks if point a dominates point b. A point a dominates b if a is cheaper or the same price and larger in size, with at least one condition being strictly true.
sequential_scan_skyline(points_list)	Implements the Sequential Scan method to find skyline points. Iterates through the dataset and adds points to the skyline if they are not dominated by any other point.
mindist_to_origin(mbr)	Calculates the minimum squared distance from the origin (0, 0) to the MBR of a node. Used for sorting nodes in the BBS algorithm.
bbs_skyline_search(rtree)	Implements the Best-Bound Skyline (BBS) algorithm using an R-tree structure to find skyline points. Traverses nodes, checks data points, and updates the skyline list based on domination checks.
divide_dataset(points_list, dimension='x')	Splits the dataset into two subspaces based on a chosen dimension (x or y). Used for the Divide-and-Conquer approach in skyline detection.

bbs_divide_and_conquer(points_list)	Applies the BBS algorithm in a Divide-and-Conquer approach. Splits the dataset, builds separate R-trees for subspaces, finds skylines for each, and merges them, filtering out dominated points.
main()	Coordinates the reading of the dataset, execution of the skyline algorithms (Sequential Scan, BBS, and Divide-and-Conquer), measures execution times, and writes the results to an output file.

create_rtree.py

Function Name (parameters)	Description (detailed information)
main(points_list)	Constructs an R-tree using the provided list of points and returns the tree. Iteratively inserts points into the tree while maintaining the MBR and handling overflow through node splitting.
Node.__init__()	Initializes a Node instance with attributes for id, child_nodes, data_points, parent, and MBR. Prepares the node to be a leaf or internal node.
Node.perimeter()	Calculates the half-perimeter of the node's MBR. Used for determining the best subtree for data insertion.

Node.is_overflow()	Checks if the number of data points or child nodes in a node exceeds the defined maximum capacity (B). Returns True if the node overflows.
Node.is_root()	Returns True if the node is the root of the R-tree (i.e., has no parent).
Node.is_leaf()	Returns True if the node is a leaf (i.e., has no child nodes).
RTree.__init__()	Initializes the R-tree by creating an empty root node.
RTree.insert(u, p)	Inserts a data point p into the R-tree starting from node u. Updates MBRs and handles node overflow to ensure efficient spatial indexing.
RTree.choose_subtree(u, p)	Selects the best subtree for inserting a data point by minimizing the increase in the perimeter of the MBR. Returns the selected child node.
RTree.peri_increase(node, p)	Calculates the increase in perimeter of a node's MBR when a new data point is added. Helps decide the optimal insertion path.
RTree.handle_overflow(u)	Splits an overflowing node into two nodes and restructures the tree as needed, creating a new root if

	necessary. Maintains tree balance.
RTree.split(u)	Splits an overflowing node u into two new nodes, minimizing the combined perimeter sum. Updates parent references for the newly created nodes.
RTree.add_child(node, child)	Adds a child node to a parent node and updates the parent's MBR accordingly.
RTree.add_data_point(node, data_point)	Adds a data point to a leaf node and updates the node's MBR.
RTree.update_mbr(node)	Updates the MBR of a node to accurately cover all its child nodes or data points, ensuring proper spatial representation.

5. Analyzing the Working of Task 1

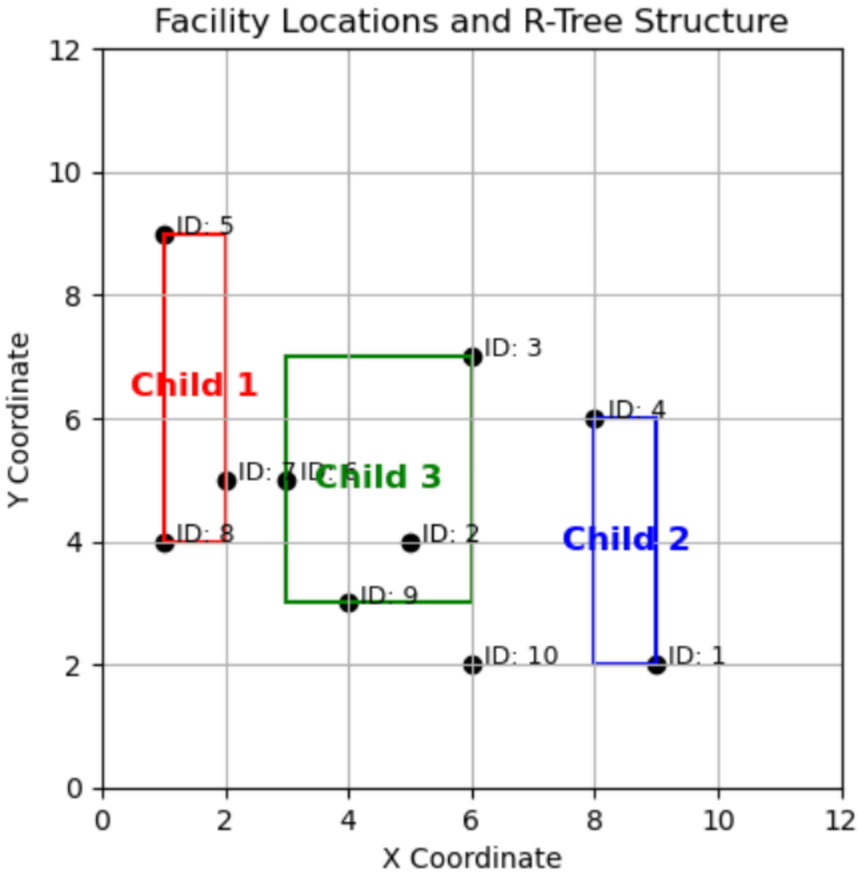
5.1 R-Tree Construction

R-Tree Construction

Process of constructing an R-Tree with the given facility location points:

1. Insert ID 1 (9,2): Insert into the root node.
2. Insert ID 2 (5,4): Add to the root node.
3. Insert ID 3 (6,7): Add to the root node.

4. Insert ID 4 (8,6): Add to the root node.
5. Insert ID 5 (1,9): The root node is full; perform a split.
    - Split into two groups:
        - Group 1: (1,9), (5,4), (6,7)
        - Group 2: (8,6), (9,2)
6. Insert ID 6 (3,5): Choose the best group based on minimum bounding rectangle (MBR) increase. Group 1 is chosen.
7. Insert ID 7 (2,5): Choose Group 1 based on the MBR increase.
8. Insert ID 8 (1,4): Choose Group 1. Now Group 1 splits:
    - New Group 1: (1,9), (2,5), (3,5)
    - New Group 3: (1,4), (5,4), (6,7)
9. Insert ID 9 (4,3): Choose Group 2, as it requires the smallest MBR increase.
10. Insert ID 10 (6,2): Choose Group 2, leading to another split:
    - New Group 2: (8,6), (9,2)
    - New Group 4: (4,3), (6,2)



5.2 BF Algorithm

Best-First (BF) Algorithm Process

With the sample data provided, the R-Tree has 3 child nodes: Child 1, Child 2, and Child 3. The Best-First algorithm starts by calculating the minimum distance from the query point (4,7) to the Minimum Bounding Rectangle (MBR) of each child node.

Step 1: Calculate Distances to MBRs

The root node is passed to the tree_traversal function, which identifies the 3 child nodes. It then begins calculating the distances between the query point and each child node's MBR.

- **Query Point**: x=4, y=7

Child 1

- MBR Coordinates: x1=1, x2=2, y1=4, y2=9

- Since y (7) is within the range of y1 and y2, only the x-distance is considered.

- Calculate the minimum x-distance:
    - Distance=|x−x1|=|4−1|=3
    - Distance=|x−x2|=|4−2|=2

- **Minimum Distance to Child 1 MBR**: 2.00

Child 2

- MBR Coordinates: x1=6, x2=9, y1=2, y2=6
- Both x and y of the query point are outside the MBR.
- Calculate minimum distances in both x and y:
    - x distance: min(|4−6|,|4−9|)=2
    - y distance: min(|7−2|,|7−6|)=1
- Use the Euclidean distance formula:
    - Distance= sqrt{(2^2 + 1^2)} = sqrt{4 + 1} = 2.24

Minimum Distance to Child 2 MBR: 2.24

Child 3

- MBR Coordinates: x1=4, x2=6, y1=4, y2=7
- x and y of the query point are inside this MBR.
- **Minimum Distance to Child 3 MBR**: 0

Step 2: Choose the Closest Child Node

Since Child 3 has the smallest distance (0), it is selected as the next node to explore. Since Child 3 is a leaf node, we then evaluate each data point within it to determine the closest point to the query point.

Step 3: Calculate Distances to Data Points in Child 3

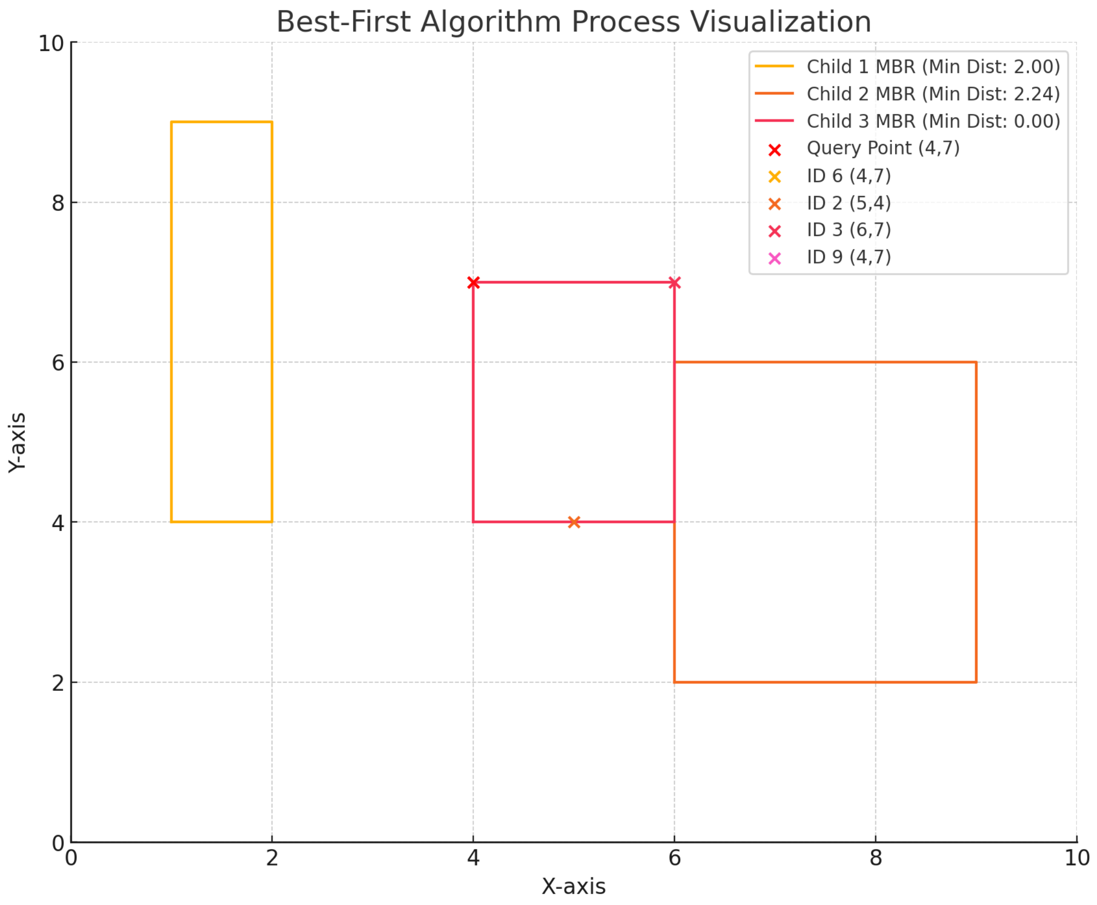
The Euclidean distance from the query point (4,7) to each data point in Child 3 is calculated:

1. **ID 6**:
    - Coordinates: x=4, y=7
    - Distance: (4−4)2+(7−7)2=0\sqrt{(4 - 4)^2 + (7 - 7)^2} = 0(4−4)2+(7−7)2=0
2. **ID 2**:
    - Coordinates: x=5, y=4
    - Distance: (4−5)2+(7−4)2=1+9=3.16\sqrt{(4 - 5)^2 + (7 - 4)^2} = \sqrt{1 + 9} = 3.16(4−5)2+(7−4)2=1+9=3.16
3. **ID 3**:
    - Coordinates: x=6, y=7
    - Distance: (4−6)2+(7−7)2=4=2\sqrt{(4 - 6)^2 + (7 - 7)^2} = \sqrt{4} = 2(4−6)2+(7−7)2=4=2
4. **ID 9**:
    - Coordinates: x=4, y=7
    - Distance: (4−4)2+(7−7)2=0\sqrt{(4 - 4)^2 + (7 - 7)^2} = 0(4−4)2+(7−7)2=0

Step 4: Determine the Closest Point

Since there are two points with a distance of 0 (ID 6 and ID 9), the algorithm returns the first occurrence, which is **ID 6**.

This will trigger the exit of the algorithm, with **ID 6** as the nearest neighbor to the query point (4,7).



5.3 Divide-and-Conquer

Divide and Conquer Process

With the sample data provided, the Divide and Conquer process for finding the nearest neighbor begins by splitting the dataset based on the x-coordinate values. Here's a step-by-step breakdown:

Step 1: Sort Data by X-Coordinates

The facility location data points are sorted based on the x-coordinate:

1. ID 5: (x=1, y=9)
2. ID 8: (x=1, y=4)
3. ID 7: (x=2, y=5)
4. ID 6: (x=4, y=7)
5. ID 9: (x=4, y=7)
6. ID 2: (x=5, y=4)
7. ID 3: (x=6, y=7)
8. ID 10: (x=6, y=2)
9. ID 4: (x=8, y=6)
10. ID 1: (x=9, y=2)

Step 2: Split the Dataset

Calculate the midpoint as the total length of the dataset divided by 2. In this case, midpoint=10/2=5\text{midpoint} = 10 / 2 = 5midpoint=10/2=5. Using the midpoint, split the dataset into two subsets: Left and Right.

- **Left Subset**:
    - ID 5: (x=1, y=9)
    - ID 8: (x=1, y=4)
    - ID 7: (x=2, y=5)
    - ID 6: (x=4, y=7)
    - ID 9: (x=4, y=7)
- **Right Subset**:
    - ID 2: (x=5, y=4)
    - ID 3: (x=6, y=7)
    - ID 10: (x=6, y=2)
    - ID 4: (x=8, y=6)
    - ID 1: (x=9, y=2)

Step 3: Construct R-Trees for Each Subset

Using each subset, construct R-Trees to facilitate efficient search:

- **Left Tree** contains points with IDs 5, 8, 7, 6, and 9.
- **Right Tree** contains points with IDs 2, 3, 10, 4, and 1.

Step 4: Perform Nearest Neighbor Search in Each Tree

With the query point (4,7), perform the nearest neighbor search in each subset independently:

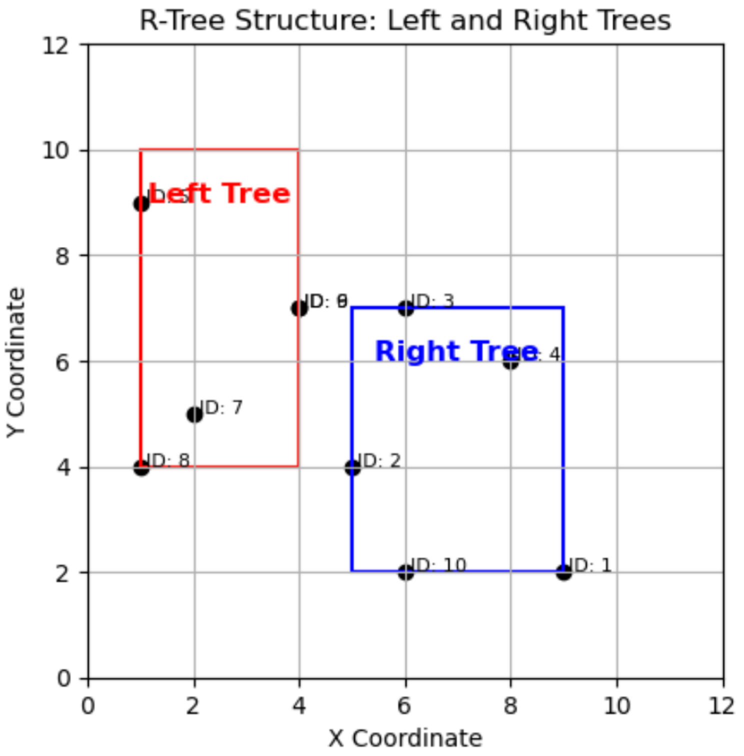
1. **Left Tree**:
    - **Calculate Distances**:
        - ID 6: (x=4, y=7), Distance = 0 (exact match)
        - ID 9: (x=4, y=7), Distance = 0 (exact match)
        - Other points in Left Tree have larger distances.
    - **Result**: The closest point in the Left Tree is ID 6 (distance = 0).
2. **Right Tree**:
    - **Calculate Distances**:
        - ID 3: (x=6, y=7), Distance = 2
        - Other points in Right Tree have larger distances.
    - **Result**: The closest point in the Right Tree is ID 3 (distance = 2).

Step 5: Comparing Results from Both Trees

Compare the nearest points found in each subset:

- The Left Tree returned a distance of 0 (ID 6).
- The Right Tree returned a distance of 2 (ID 3).

Since the Left Tree's closest point (ID 6) has a smaller distance (0), it is confirmed as the nearest neighbor to the query point (4,7).



6. Analyzing the Working of Task 2

6.1 R-Tree Construction

Step 1: Insert ID 1 into the root node
Root:

- Data Points = [(5, 10)]

Step 2: Insert ID 2 into the root node
Root:

- Data Points = [(5, 10), (8, 7)]

Step 3: Insert ID 3 into the root node
Root:

- Data Points = [(5, 10), (8, 7), (12, 5)]

Step 4: Insert ID 4 into the root node
Root:

- Data Points = [(5, 10), (8, 7), (12, 5), (3, 12)]

Step 5: Insert ID 5 into the root node

Root:

- Data Points = [(5, 10), (8, 7), (12, 5), (3, 12), (10, 8)]
- Overflow triggered, split is called to reduce the size

End of step:

- Child 1 = [(3, 12), (5,10)]
- Child 2 = [(8, 7), (10, 8), (12, 5)]

Step 6: Insert ID 6 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2]
- Identifies Child 2 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5, 10)]
- Child 2 = [(8, 7), (10, 8), (12, 5), (6, 6)]

Step 7: Insert ID 7 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 2 as the best child based on the perimeter increase
- Child 2 = [(8, 7), (10, 8), (12, 5), (6, 6), (9, 4)]
- This triggers the overflow
- Split is called to reduce the child

End of step:

- Child 1 = [(3, 12), (5,10)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5)]

Step 8: Insert ID 8 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 1 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5, 10), (2, 14)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5)]

Step 9: Insert ID 9 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 3 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5, 10), (2, 14)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5), (13, 3)]

Step 10: Insert ID 10 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 3 as the best child based on the perimeter increase
- Child 3 = [(9, 4), (10, 8), (12, 5), (13, 3), (15, 2)]
- This triggers the overflow
- Split is called to reduce the child

End of step:

- Child 1 = [(3, 12), (5,10), (2, 14)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5)]
- Child 4 = [(13, 3), (15, 2)]

Step 11: Insert ID 11 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 1 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5,10), (2, 14), (4, 9)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5)]
- Child 4 = [(13, 3), (15, 2)]

Step 12: Insert ID 12 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 3 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5,10), (2, 14), (4, 9)]
- Child 2 = [(6, 6), (8, 7)]
- Child 3 = [(9, 4), (10, 8), (12, 5), (11, 6)]
- Child 4 = [(13, 3), (15, 2)]

Step 13: Insert ID 13 into the root node

Root:

- Root Child Nodes = [Child 1, Child 2, Child 3]
- Identifies Child 2 as the best child based on the perimeter increase

End of step:

- Child 1 = [(3, 12), (5,10), (2, 14), (4, 9)]
- Child 2 = [(6, 6), (8, 7), (7, 3)]
- Child 3 = [(9, 4), (10, 8), (12, 5), (11, 6)]
- Child 4 = [(13, 3), (15, 2)]

Step 14: Insert ID 14 into the root node

Root:

- Root Child Nodes = [Group 1, Group 2]
- Identifies Child 1 as the best child based on the perimeter increase
- Child 1 = [(3, 12), (5,10), (2, 14), (4, 9), (1, 13)]
- This triggers the overflow
- Split is called to reduce the child

End of step:

Group 1:

- Child 1 = [(1, 13), (3, 12), (2, 14)]
- Child 2 = [(4, 9), (5,10)]

Group 2:

- Child 3 = [(6, 6), (8, 7), (7, 3)]
- Child 4 = [(9, 4), (10, 8), (12, 5), (11, 6)]
- Child 5 = [(13, 3), (15, 2)]

Step 15: Insert ID 15 into the root node

Root:

- Root Child Nodes = [Group 1, Group 2]
- Identifies Child 5 as the best child based on the perimeter increase

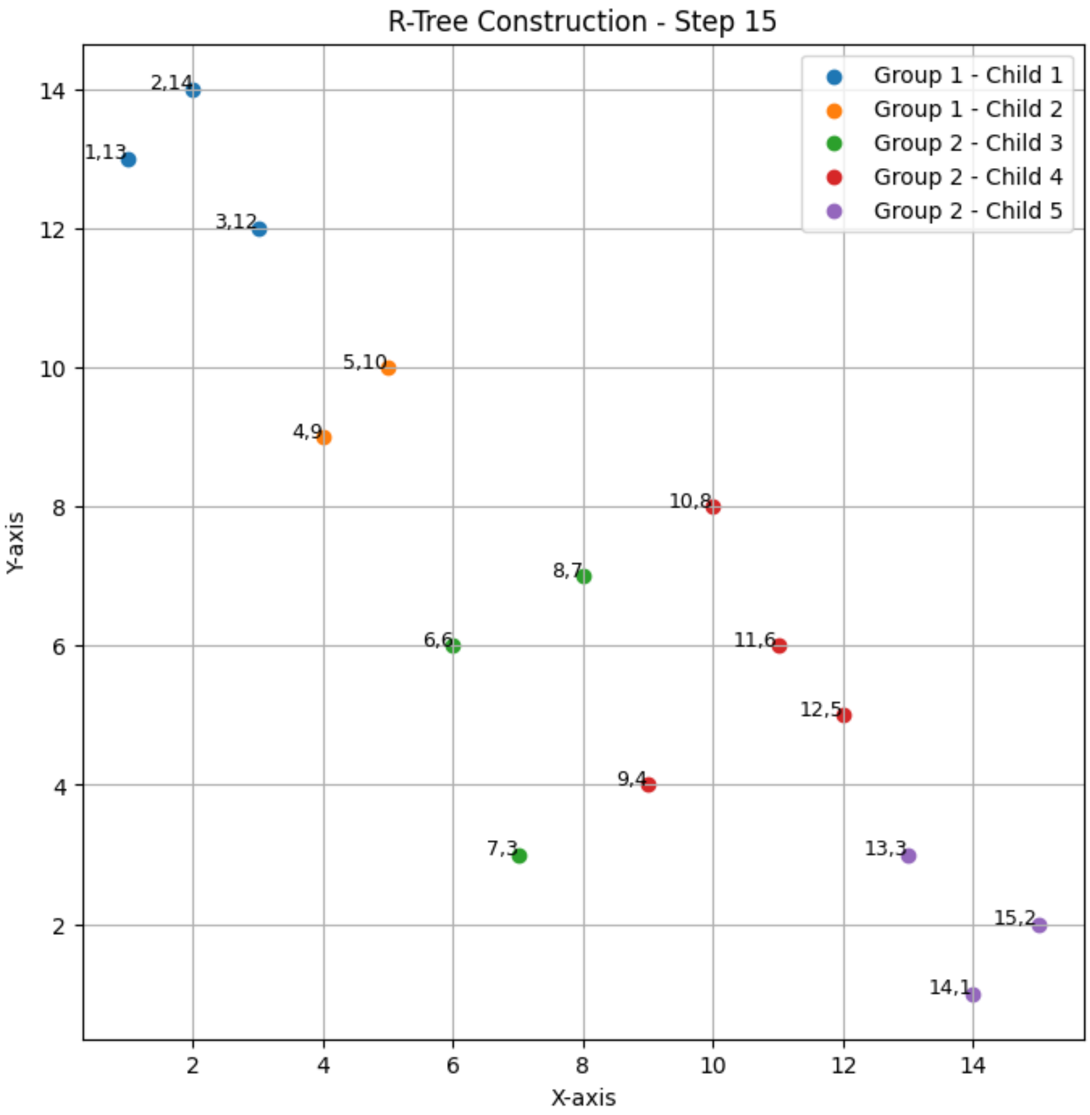
End of step:

Group 1:

- Child 1 = [(1, 13), (3, 12), (2, 14)]
- Child 2 = [(4, 9), (5,10)]

Group 2:

- Child 3 = [(6, 6), (8, 7), (7, 3)]
- Child 4 = [(9, 4), (10, 8), (12, 5), (11, 6)]
- Child 5 = [(13, 3), (15, 2), (14, 1)]



6.2 BBS Algorithm

BBS Algorithm Process

Given the R-tree structure with 5 child nodes across 2 groups, the Best-First Search (BBS) algorithm will calculate the minimum distance from a query point (5,5) to the Minimum Bounding Rectangle (MBR) of each child node, using the closest nodes to traverse further.

Step 1: Calculate Distances to MBRs

The algorithm starts by calculating distances from the query point to each child node's MBR.

- Query Point: x=5 y=5

Group 1

Child 1

- MBR Coordinates: x1=1, x2=3, y1=12, y2=14
- Both x and y of the query point are outside the MBR.
- Calculate the minimum distances in both x and y:
    - x distance: min(|5−1|,|5−3|)=2

    - y distance:min(|5−12|,|5−14|)=7

- Use the Euclidean distance formula:
    - Distance: sqrt{(2^2 + 7^2)} = sqrt{4 + 49} = 7.28

- Minimum Distance to Child 1 MBR: 7.28

Child 2

- MBR Coordinates: x1=4, x2=5,y1=9,y2=10
- Both x and y of the query point are outside the MBR.
- Calculate minimum distances in both x and y:
    - x distance: min(|5−4|,|5−5|)=0
    - y distance: min(|5−9|,|5−10|)=4
- Use the Euclidean distance formula:
    - Distance: sqrt{(0^2 + 4^2)} = sqrt{16}=4
- Minimum Distance to Child 2 MBR: 4.00

Group 2

Child 3

- MBR Coordinates: x1=6, x2=8, y1=3, y2=7
- X and y are partially within the MBR range.
- Calculate minimum distances in x and y:
    - x distance: |5−6|=1
    - y distance: 0 (since y=5 is within the MBR's y-range)
- Distance: 1
- Minimum Distance to Child 3 MBR: 1.00

Child 4

- MBR Coordinates: x1=9, x2=12, y1=4, y2=8
- Both x and y are outside the MBR.
- Calculate minimum distances in both x and y:
    - x distance: min(|5−9|,|5−12|)=4
    - y distance: min(|5−4|,|5−8|)=1
- Use the Euclidean distance formula:
    - Distance: sqrt{(4^2 + 1^2)} = sqrt{16 + 1} = 4.12
- Minimum Distance to Child 4 MBR: 4.12

Child 5

- MBR Coordinates: x1=13, x2=15, y1=1, y2=3
- Both x and y are outside the MBR.
- Calculate minimum distances in both x and y:
    - x distance: min(|5−13|,|5−15|)= 8
    - y distance: min(|5−1|,|5−3|)=2
- Use the Euclidean distance formula:

    - Distance: sqrt{(8^2 + 2^2)} = sqrt{64 + 4} = 8.25
- Minimum Distance to Child 5 MBR: 8.25

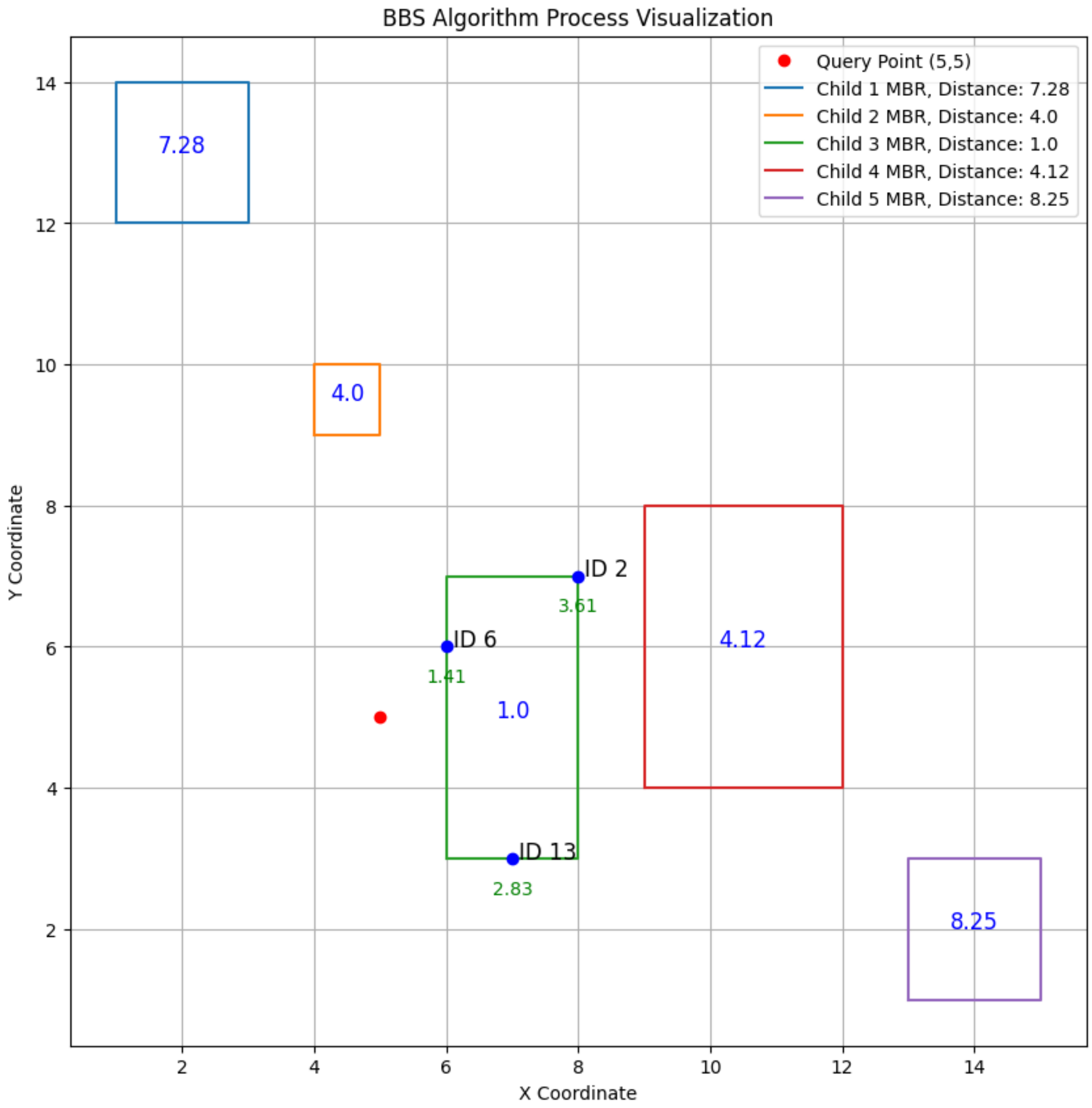
Step 2: Choose the Closest Child Node
After calculating the distances to each MBR, the algorithm identifies Child 3 as the closest child node to the query point with a distance of 1.00. Therefore, Child 3 is selected as the next node to explore.

Step 3: Calculate Distances to Data Points in Child 3
Since Child 3 is a leaf node, we now evaluate each data point within it to determine the closest point to the query point (5,5).

1. ID 6:
    - Coordinates: x=6, y=6
    - Distance: sqrt{(5 - 6)^2 + (5 - 6)^2} = sqrt{1 + 1} = 1.41
2. ID 2:
    - Coordinates: x=8, y=7
    - Distance: sqrt{(5 - 8)^2 + (5 - 7)^2} = sqrt{9 + 4} = 3.61
3. ID 13:
    - Coordinates: x=7, y=3
    - Distance: sqrt{(5 - 7)^2 + (5 - 3)^2} = sqrt{4 + 4} = 2.83

Step 4: Determine the Closest Point

The closest point to the query point (5,5) within Child 3 is ID 6 with a distance of 1.41.



6.3 Divide-and-Conquer

Given Data

The R-tree structure is:

- Group 1:
  - Child 1: (1,13),(3,12),(2,14)(1, 13), (3, 12), (2, 14)(1,13),(3,12),(2,14)
  - Child 2: (4,9),(5,10)(4, 9), (5,10)(4,9),(5,10)
- Group 2:

- Child 3: (6,6),(8,7),(7,3)(6, 6), (8, 7), (7, 3)(6,6),(8,7),(7,3)
- Child 4: (9,4),(10,8),(12,5),(11,6)(9, 4), (10, 8), (12, 5), (11, 6)(9,4),(10,8),(12,5),(11,6)
- Child 5: (13,3),(15,2),(14,1)(13, 3), (15, 2), (14, 1)(13,3),(15,2),(14,1)

Divide and Conquer Approach

We want to find the point nearest to a query point, say (5,5) using the Divide and Conquer approach.

Step 1: Divide the R-tree into Regions

The R-tree is already divided into groups and children, which we'll treat as spatially grouped regions.

- Group 1 with Child 1 and Child 2
- Group 2 with Child 3, Child 4, and Child 5

These groups are pre-divided regions, so we don't need additional division.

Step 2: Determine the Bounding Box (MBR) for Each Group

1. Group 1 Bounding Box (MBR):
    - Minimum x: 1 (from Child 1)
    - Maximum x: 5 (from Child 2)
    - Minimum y: 9 (from Child 2)
    - Maximum y: 14 (from Child 1)
2. Group 2 Bounding Box (MBR):
    - Minimum x: 6 (from Child 3)
    - Maximum x: 15 (from Child 5)

    - Minimum y: 1 (from Child 5)
    - Maximum y: 8 (from Child 4)

Step 3: Calculate Distance to Each Group's MBR from the Query Point

Using the query point (5,5), calculate the distance to the bounding box of each group.

- Distance to Group 1 MBR:
    - x-distance: 0 (since 5 is within x-range of 1 to 5)
    - y-distance: min(∣5−9∣,∣5−14∣)=4
    - Distance: 4
- Distance to Group 2 MBR:
    - x-distance: ∣5−6∣=1 (since 5 is just outside the x-range)
    - y-distance: 0 (since 5 is within y-range of 1 to 8)
    - Distance: 1

Since Group 2 has the smaller distance (1), we’ll prioritize it for further exploration.

Step 4: Divide Group 2 into Children and Calculate Distances to Each Child's MBR

Now that we know Group 2 is closer, we’ll focus on its children.

1. Child 3 MBR:
    - Min x: 6, Max x: 8
    - Min y: 3, Max y: 7
    - Distance to Query Point (5,5):
        - x-distance: ∣5−6∣=1
        - y-distance: 0 (since 5 is within y-range)
        - Distance: 1

2. Child 4 MBR:
   - Min x: 9, Max x: 12
   - Min y: 4, Max y: 8
   - Distance to Query Point (5,5):
     - x-distance: |5−9|=4|
     - y-distance: 0
     - Distance: 4
3. Child 5 MBR:
   - Min x: 13, Max x: 15
   - Min y: 1, Max y: 3
   - Distance to Query Point (5,5):
     - x-distance: |5−13|=8
     - y-distance: |5−3|=2
     - Distance:sqrt{(8^2 + 2^2)} = 8.25

Child 3 has the smallest distance (1), so we'll explore Child 3 further.

Step 5: Evaluate Points in Child 3 for Closest Point to Query Point

Now that we're focused on Child 3, we calculate the Euclidean distance from the query point (5,5) to each point in Child 3.

1. Point (6,6):
   - Distance: sqrt{(5 - 6)^2 + (5 - 6)^2} = sqrt{1 + 1} = 1.41
2. Point (8,7):
   - Distance: sqrt{(5 - 8)^2 + (5 - 7)^2} = sqrt{9 + 4} = 3.61

3. Point (7,3):

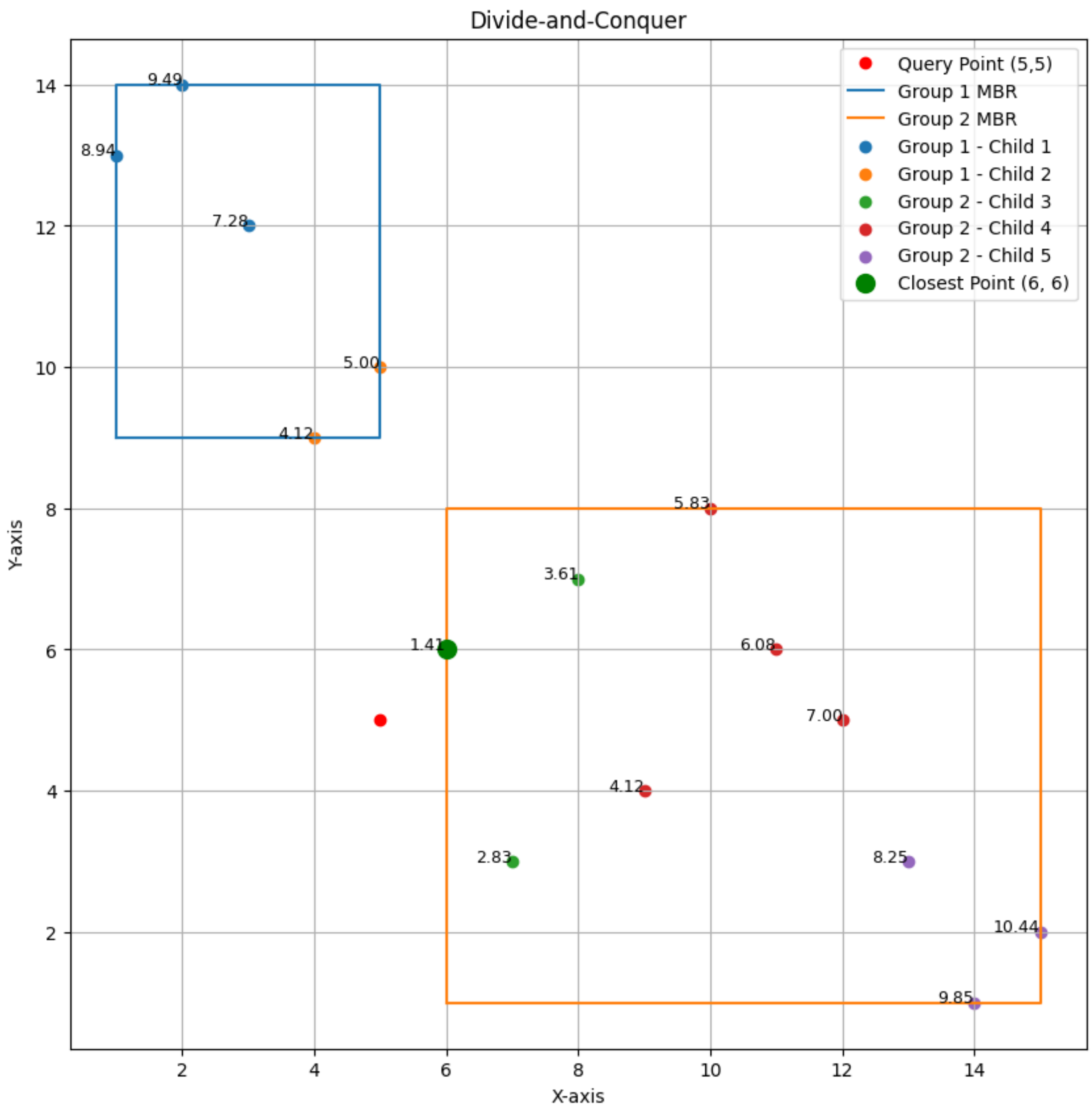
   - Distance: sqrt{(5 - 7)^2 + (5 - 3)^2} = sqrt{4 + 4} = 2.83

Step 6: Determine the Closest Point

The closest point in Child 3 to the query point (5,5) is Point (6,6) with a distance of 1.41.

Conclusion

The Divide and Conquer approach terminates with Point (6,6) as the nearest neighbor to the query point (5,5).



Video Presentation URL:

https://drive.google.com/file/d/1X_emUTL5RpSel_yf7hZepjiltIfzWa3R/view?usp=sharing