**INFS7205/4205 Report for Assignment 1**

*(s4358277\_Clarke)*

# 1 Program execution requirements

## 1.1 Program environment

Operating System: Windows 10.

Programing Language: Java 8. jre1.8.0\_161

Eclipse Neon.3

## 1.2 Input files and parameters

**Usage:** datasetLocation queryLocation saveLocation m

**datasetLocation:** file path for the dataset to insert into the tree

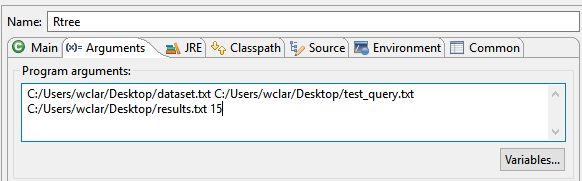
**queryLocation:** file path of test queries file

**saveLocation:** file path of where to save the results of the queries (include filename)

**m:** How many points to store in each node. Best performance seems to be about 15. For whatever reason setting m=5 will cause a stack overflow error. No other value from 3-20 will cause this.

**Example:**

C:/Users/wclar/Desktop/dataset.txt C:/Users/wclar/Desktop/test\_query.txt C:/Users/wclar/Desktop/results.txt 15

In Eclipse go to Run Configuration -> Arguments and then write the parameters there with space between each.

## 1.3 Other requirements

Eclipse -> Import -> Existing Projects into workspace -> select folder R-tree as root directory and import.

Run inside eclipse using Run Configurations to input parameters.

# 2 Program documentation

## 2.1 Program organisation

|  |  |
| --- | --- |
| **Class/File Name** | **Description** |
| Rtree.java | File that implements the Rtree data structure and methods to insert and search the tree. Also contains methods for load and saving files. |
| Node | A node class was created for the underlying data structure of the tree. A Node stores; its MBR, Boolean describing if it is a leaf, its parent node, and Linked List of its children. |
| Entry (Subclass of Node) | Entry extends node and stores the ID of a data point. It is attached to a Node as its leaf |

## 

## 2.2 Function description

Unless the function is a Trivial function, please include the function name as well as a brief description of the function goal, and a description of the function itself. Trivial functions include getters, setters, constructors, and simple calculations like *GetPerimeter (Point bottomLeft, Point topRight)*.

|  |
| --- |
| **Node.addLeafNode(Entry e)** |
| Adds an Entry e to a Leaf Node. i.e. adding data points to a leaf. Updates its MBR with each new point added. Used when adding entries to a leaf node |
| Add the entry to a Linked List of entries in the Node. Set e’s parent to this node. Then update the nodes MBR by looking up the (x,y) coordinates for each id and retrieving the min x and y values and max x and y values. |
| **Node.addNode(Node n)** |
| Adds a Node n to a Node. i.e. adding internal nodes. Updates its MBR with each new point added. Used when adding leaf nodes to the tree. Used when adding Non-leaf nodes. |
| Adds node to the Linked List of Children of Node. Sets n’s parent to this node. Then update the nodes MBR by looking up the MBR of each child node and determining the new MBR |
| **Rtree.insert(Node n, int id)** |
| Insert the id into the r-tree. Initially invoked as Rtree.insert(root, id); |
| If n is a leaf node, then we make a new entry using id and add the entry to n. Check if this causes an overflow and handle it. Otherwise, n is not a leaf node and we must choose a suitable child of n and try inserting id into that child. Continues recursively till we find a leaf node. |
| **Rtree.rangeQuery(Node n, int[] range)** |
| Find all the points in the Rtree that fit within the range. Return number of points found. Initially invoked as Rtree.rangeQuery**(root n, int[] range)** |
| If n is a leaf, then we search through the data points and count any that fit inside range. Otherwise, perform the same range query on the children of n whose MBR intersect range. Recursive call until all suitable leaves are reached. |
| **handleOverflow(Node n)** |
| Handles an overflowing node by splitting the node, creating a new node and inserting these splits into the new node. |
| Split n into u and udash (different add method is called depending on whether n is a leaf node). If n was the root, create a new root with u and udash as its children. Otherwise, create a new node w and add u and udash to it. Add to w to the parent of n if it overflows handle overflow(n.parent). Recursive Call till no more overflows. |
| **splitNode(Node n)** |
| Return recommended split for Node n |
| Split the node by following the heuristics described in lecture slide. Very similar operation for leaf or non-leaf nodes. Generate a list L which sorts the points in n on x (leaf) or sorts MBRs in n by their lower x dimension(non-leaf). For I = ceiling of (0.4\*m) to (number points in n – ceil of (0.4\*m)). S1 is the first I points or rectangles in n, s2 is the rest. Calculate the perimeter sum of MBR(s1) and MBR(s2) record the best split. Repeat w.r.t y for leaf nodes and repeat w.r.t. right boundary on x dimension and then the y dimension for non-leaf nodes. |