**ITEC874/COMP733 Report for Assignment 2**

*45786690\_GOLI*

1. **Program Execution Requirements**
   1. **Program Environment-** The algorithm is coded in python and run in python version 3.5.2 environment and windows operating system. For testing and editing purposes, I have also executed the code in jupyter notebook and observed the efficiency and results of the code in the same environment. The Jupyter Notebook is an open-source web application that facilitates a programmer to create python documents containing live data, formulas, visualizations and lets us to add narrative text describing the results and modules of the code. Due to limitations in performance of my Personal computer I have executed r tree algorithm in a GPU equipped environment as well to gauge the peak efficiency of my algorithm. Google Colab is a free online cloud services platform which lets a programmer to run code in a GPU run jupyter environment.
      1. **Input files and parameter-**

* “Dataset for R-Tree.txt”- Dataset for R-Tree.txt is an input text file consisting of raw data points in [x,y] format. It consists of 100000 point instances which is used to construct a r tree.
* “range\_queries.txt”- This file consists of 100 range queries in the format of [x1,x2,y1,y2].
* “xx.py”- This python file consists of source code of r tree algorithm.
  + - 1. **Other requirements-** The source includes following modules:
* Sys - The sys module provides information about constants, functions and methods of the Python interpreter. I have used this module to retrieve the command line arguments and use it in the code. Sys.argv is an array or list of arguments specified by the programmer through command line. The input and test\_cases file is fed into the code through CLI(command line interface).
* Time, tqdm - The algorithm also uses time module to approximately measure the time complexity of the algorithm and further gauge its efficiency. The efficiency of sequential search and r tree algorithm can be compared using this module. In addition to time, I have imported tqdm module to track and monitor the progress of the algorithm.
* Pandas - The input is fed to code in form of a raw text file which consists of many whitespaces and unnecessary characters. To read, manipulate and properly organize that information, the code uses the pandas library. Pandas library is used in my code for data manipulation and cleaning.

**2.Program documentation**

**2.1 Program organisation**

If your assignment involves multiple files and/or classes, please include brief, high level descriptions of each file/class in your program as shown below.

|  |  |
| --- | --- |
| **Class/File Name** | **Description** |
|  |  |
| Rtree | The code is properly organized and well presented in form of classes and modules. This Algorithm consists of two classes- Rtree and node, mimicking the real time functionality and properties of the R Tree. The rtree class is a crucial class which has necessary functions like search(),node\_split(),etc included in it, which is responsible for construction and management of the tree. The class has root as the only attribute and has the following functions in it:   * Search() * check\_intersections() * insert() * node\_split() * optimal\_node\_split() * insert\_child() * add\_points() * update\_min\_bound() |
|  |  |
|  |  |
| Node | Node is another class defined in the code indicating the nodes of the tree. The rtree class acts as an infrastructure provider to the objects of the node class which can be fit into it. Generally a node objects in other tree algorithms has many properties like value of node, parent, children etc. Likewise the Node class of this code has attributes like children(refers to the list of nodes its referring to under lower level), data points(if it is a leaf node=>it store the 2d coordinates of the points), predecessor(refers to the parent node), Min\_bound(consists of coordinates of a minimum bounding rectangle) describing the orientation of the node in 2 Dimensional space and its position in the tree. The class consists of the following modules:   * get\_perimeter()- calculates the perimeter of the Minium bounding rectangle of that node * overflow\_report()- Checks whether the node is struck at overflow condition * check\_root\_condition()- checks whether the node is root * check\_leaf\_condition()- checks whether the node is leaf |
|  |  |
|  |  |
|  |  |

* 1. **Function description**

|  |  |
| --- | --- |
| **Function Name (parameters)** | **Description** |
| **sequential\_search\_query(points,search\_query)** | Sequential search query is a simple linear search algorithm which exhaustively iterates through each data point in the dimensional space and returns the number of points that lie in the given range query.  **Parameters:**   * Points- set of all data points present in r tree * Search\_query- range query that has to be searched |
| **get\_maximum (value\_1,value\_2)** | This module does comparison of two values that are fed into it and returns the maximum of the two values. |
| **get\_minimum(value\_1,value\_2)** | The module get\_minimum() compares two values that are fed into it and returns the minimum value. |
| **insert (self, node, p)** | The insert module is main wrapper to the insertion procedure of the r tree algorithm. It internally calls select\_optimal\_subtree() to traverse through optimal subtree and add data points to the leaf node.  **Parameters:**   * P- data point |
| **select\_optimal\_subtree(self, node, point)** | The select\_optimal \_subtree is module used to traverse through feasible appropriate subtree of node “node”. It is internally called by the insert() module to traverse through the tree.  **Parameters:**   * Node- a node of r tree * Point- data point |
| **insert\_child(self, parent, child)** | This module inserts the internal nodes of the tree to a particular parent node. This module assists in handling the overflow cases. The module overflow\_call() internally calls this function to assign the splits to the parent nodes. It also updates the minimum bounding rectangle of the parent to which splits are added to.  **Parameters:**   * child- a node that has to be inserted * parent- node of r tree in that traversal |
| **overflow\_call(self, node)** | This function is triggered to handle the overflow condition of the node. It internally calls the node\_split() module to bifurcate the data points or child nodes. The resultant splits are assigned to the parent. It also internally calls the overflow function to handle the overflow of the parent to which the new splits are added. After adding the splits to the parent the function calls update\_min\_bound() to change the mbr of the parent of which the two new splits are added to.  **Parameters:**   * Node- a node of r tree |
| **add\_points(self, parent, data\_point)** | The module add\_points() add the data points to the tree. This module is internally called by the wrapper insert() function to add the data points to the nodes present in the lowest bottom level of the tree. It also updates the minimum bounding rectangle of the parent to which the data points are added to. |
| **node\_split(self, node)** | The node\_split function identifies the ideal split of node ”node”.  **Parameters:**   * Node- a node of r tree |
| **optimal\_node\_split (self,sample,m,minimum\_perimeter,flag)** | The optimal\_node\_split returns the best split of the “sample” passed to this function using perimeter as a factor.  **Parameters:**   * Sample- data sample in which dimension a split has to be performed * Flag- if flag is set it is leaf |
| **update\_min\_bound(self, node)** | The update\_min\_bound() function updates the minimum bounding rectangle of the “node” passed to it.  **Parameters:**   * Node- a node of r tree |
| **search (self, node, range\_query)** | The search module iterates through the tree and returns the count of the number of points which lies in the given range\_query region. It internally calls intersection function to check whether the given data point lies in the range of the query. The recursive calls of the search function let it to iterate through the tree and traverse through all the nodes and data points of the tree.  **Parameters:**   * Node- a node of r tree * Range\_query- range query |
| **check\_intersections(self, node, query)** | Check\_intersection module checks whether the minimum bounding rectangle of the specified node intersects the minimum bounding rectangle of the range\_query or search\_query.  **Parameters:**   * Node- a node of r tree * query- range query |
|  |  |
| **changein\_perimeter(self, node, point)** | The changein\_perimeter() returns the increment in parameter if a particular point is added to a node. This module assists the wrapper insert() module in traversing the right subtree in which a data point has to be added using change in perimeter as a factor. The insertion module traverses through subtree to which the addition of data point has less effect on the perimeter of the node and change\_perimeter() module exactly calculate the increase in value of the node if point “point”is added. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Screenshots:

