COMP-6521

Advanced Database Technology and Applications

Lab Mini-Project 2

Efficient Indexing for 3D Points: Accelerating Range Queries and Nearest Neighbor Search

Submitted To: Dr. Nematollaah Shiri

Submitted By (**Group 12**) : -

Name	Student ID	Email
Manpreet Singh Rana	40227463	rsingh.manpreet@gmail.com
Firas El Rachidi	40202702	firasrachidi1@gmail.com
Chuanyan Hu	40168030	amy.chuanyanhu@gmail.com
Karandeep Singh	40197407	karansaini591@gmail.com

Introduction

In this project, we were given a dataset of 10 million points in 3D space, with real-number attributes in the range [0, 1000], stored in the relation Points (X, Y, Z). The objective was to develop an efficient index structure to speed up answering range and nearest neighbour (NN) queries.

After analysing various index structures, we decided to use the KD tree implementation to achieve our objectives. The KD tree is a binary tree structure that partitions the dataset recursively based on the median value of a specific dimension at each level. This partitioning strategy reduces the search space and improves the performance of range and NN queries.

However, as we began implementing the KD tree, we realized that building the tree using the median-based partitioning strategy required a significant number of resources, both in terms of time and memory. Due to the large size of the dataset, it was not practical to use the standard KD tree implementation. Therefore, we decided to adopt an ad hoc approach and used a simpler partitioning strategy to build the KD tree. This approach still improved query performance significantly compared to a brute-force approach, while being more feasible to implement given the available resources.

Program Description

Our implementation anchors on three main functions namely buildTree, rangeQuery and findNearestNeigbors. Below is brief explanation of what each function does.

a) buildTree (List<double[]> points)

1. The buildTree function takes in a list of double arrays representing the points in the dataset.

2. It initializes the root of the KD tree as null.

3. For each point in the dataset, it calls the `insert` function to add the point to the KD tree.

4. After all points have been inserted, the buildTree function returns the root of the KD tree.

The `insert` function:

1. Takes in a node in the KD tree, a double array representing a point in the dataset, and the current depth of the KD tree.

2. If the current node is null, it creates a new node with the given point and returns it.

3. Otherwise, it calculates the axis to partition the dataset based on the current depth of the tree and compares the point value in that axis with the node value in that axis.

4. If the point value is less than the node value, it recursively calls the insert function with the left child node and updates the depth.

5. Otherwise, it recursively calls the insert function with the right child node and updates the depth.

6. It returns the updated node.

b) **rangeQuery(Node node, double[] min, double[] max, List<double[]> result, int depth)**

1. The rangeQuery function takes in five parameters: a Node object representing the current node in the KD tree, two double arrays representing the minimum and maximum ranges for the query, a List of double arrays to store the results of the query, and an integer representing the current depth of the tree.

2. The function first checks if the current node is null and returns if it is.

3. The function calculates the current axis for the node by taking the depth modulo K.

4. If the point represented by the node is within the range specified by the minimum and maximum arrays for the current axis, the function checks if the point is within the full range specified by the minimum and maximum arrays for all three axes. If it is, the point is added to the results list. The function then recursively calls itself on the left and right subtrees of the current node, incrementing the depth by 1.

5. If the point represented by the node is to the left of the range specified by the minimum and maximum arrays for the current axis, the function recursively calls itself on the right subtree of the current node, incrementing the depth by 1.

6. If the point represented by the node is to the right of the range specified by the minimum and maximum arrays for the current axis, the function recursively calls itself on the left subtree of the current node, incrementing the depth by 1.

7. The function continues to recursively call itself until all nodes within the specified range have been added to the results list

c) **`findNearestNeighbors(double[] queryPoint)`**

1. The function findNearestNeighbors takes a query point as input and returns a list of the nearest neighbors of the query point.

2. It initializes two empty lists: nearest and result.

3. It then calls the `nearestNeighbor` function on the root node of the KD tree, passing in the query point and the nearest list.

4. The `nearestNeighbor` function recursively traverses the KD tree, starting at the root node and checking each node to see if it is closer to the query point than the current nearest neighbor(s).

5. If the current node is closer than the current nearest neighbor(s), the nearest list is updated accordingly.

6. If the current node is equidistant to the current nearest neighbor(s), it is added to the nearest list.

7. The `nearestNeighbor` function continues to traverse the KD tree until it has explored all relevant branches.

8. Once the `nearestNeighbor` function has finished, the result list is populated with the coordinates of the nodes in the nearest list, and this list is returned by **`the findNearestNeighbors`** function.

Calculations used for evaluating size of index created

In our tree implementation each node consists of a double array which consists of three double values. On top of that each node (except the leaf nodes) will have two pointers.

Assuming a 64-bit JVM, the memory usage of a double array of size 3 in Java would be:
Overhead for array object: 12 bytes.
Size of each double element: 8 bytes.
Total memory usage: 12 bytes (overhead) + 3 * 8 bytes (size of each element) = 36 bytes.

In addition to this, 2 pointers of size 8 bytes each, 2* 8 = 16 bytes

Therefore, size of an internal node = 36+16 = 52 bytes, whereas size of a leaf would be 36 bytes only.

In our code, index size is calculated as **`Number of Nodes * 36 + (TotalNodes - LeafNodes) * 16 bytes.`**

Results

Dataset1 – coordinates.csv

Number of data points	**10000**
Size of Index	**0.4449462890625 MB**
Time taken to build the tree	6 ms
Time taken for Range query 1(where Range includes all the data points in the tree) *(0, 0, 0) to (1000, 1000, 1000)*	3 ms
Number of output tuples returned for Range query 1	10000
Time taken for Range query 2 (where Range includes some random number of data points in the tree). *(550, 550, 550) to (1000, 1000, 1000)*	2 ms
Number of output tuples returned for Range query 2	888
Time taken for finding Nearest Neigbor of a random data point *(56.5, 86.04, 71.51)*	3 ms
Number of tuples in the output for Nearest Neigbor	1

Dataset2 – coordinates_demo.csv

Number of data points	**10000000**
Size of Index	**445.0585021972656 MB**
Time taken to build the tree	40264 ms
Time taken for Range query 1(where Range includes all the data points in the tree) *(0, 0, 0) to (1000, 1000, 1000)*	1399 ms
Number of output tuples returned for Range query 1	10000000
Time taken for Range query 2 (where Range includes some random number of data points in the tree). *(550, 550, 550) to (1000, 1000, 1000)*	256 ms
Number of output tuples returned for Range query 2	910184
Time taken for finding Nearest Neigbor of a random data point *(56.5, 86.04, 71.51)*	1 ms
Number of tuples in the output for Nearest Neigbor	1

Dataset3 – coordinates_large.csv

Number of data points	**10000000**
Size of Index	**445.0747375488281 MB**
Time taken to build the tree	30811 ms
Time taken for Range query 1(where Range includes all the data points in the tree) *(0, 0, 0) to (1000, 1000, 1000)*	1236 ms

Number of output tuples returned for Range query 1	10000000
Time taken for Range query 2 (where Range includes some random number of data points in the tree). *(550, 550, 550) to (1000, 1000, 1000)*	231 ms
Number of output tuples returned for Range query 2	910943
Time taken for finding Nearest Neigbor of a random data point *(56.5, 86.04, 71.51)*	1 ms
Number of tuples in the output for Nearest Neigbor	1

Screenshots

Dataset 1 – coordinates.csv

```
############################################################ Build Tree ##########################################################

Starting with building the tree
Tree built successfully
Time taken to build the tree: 6 milliseconds

############################################################ Tree Build Completed ##########################################################
Number of nodes 10000
```

Build Tree timings and number of Nodes.

```
############################################################ Range Query ##########################################################
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
0
Enter lower bound of y
0
Enter lower bound of z
0
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 0.0, 0.0, 0.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 3 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT1.csv having records equal to 10000
Do you want to check again for a different range (Y/N)
y
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
550
Enter lower bound of y
550
Enter lower bound of z
550
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 550.0, 550.0, 550.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 2 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT2.csv having records equal to 888
Do you want to check again for a different range (Y/N)
N
############################################################Range Query Completed ##########################################################
```

Range Query Results

```
########################################################## Nearest Neighbour Query ##########################################################################################
Enter the value of x, y and z coordinates of which you want to find the nearest neighbour for
Enter the value of x
56.5
Enter the value of y
78.5
Enter the value of z
92.5
Starting with executing NN query result for coordinates ( 56.5, 78.5, 92.5 )
NN result fetched in 3 milliseconds
NN of the entered coordinate are
[45.87, 86.04, 71.51]
Do you want to check NN again for a different coordinate (Y/N)
N
##########################################################Nearest Neighbour Query Completed ###########################################################################


########################################################## Index Size Enquiry  ###############################################################################
Size of index created is: 0.4449462890625 MB
########################################################## Program Finished  #################################################################################
```

Nearest Neighbor Query Results and Index Size Reporting

Dataset 2 – coordinates_demo.csv

```
############################################################## Build Tree ###################################################################

Starting with building the tree
Tree built successfully
Time taken to build the tree: 40264 milliseconds

############################################################## Tree Build Completed ##########################################################################
Number of nodes 10000000
```

Build Tree timings and number of Nodes.

```
############################################################## Range Query ###################################################################
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
0
Enter lower bound of y
0
Enter lower bound of z
0
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 0.0, 0.0, 0.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 1399 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT1.csv having records equal to 10000000
Do you want to check again for a different range (Y/N)
y
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
550
Enter lower bound of y
550
Enter lower bound of z
550
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 550.0, 550.0, 550.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 256 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT2.csv having records equal to 910184
Do you want to check again for a different range (Y/N)
N
##############################################################Range Query Completed ###########################################################################
```

Range Query Results

```
########################################################## Nearest Neighbour Query ##########################################################################################
Enter the value of x, y and z coordinates of which you want to find the nearest neighbour for
Enter the value of x
56.5
Enter the value of y
86.04
Enter the value of z
71.51
Starting with executing NN query result for coordinates ( 56.5, 86.04, 71.51 )
NN result fetched in 1 milliseconds
NN of the entered coordinate are
[57.05, 88.73, 71.93]
Do you want to check NN again for a different coordinate (Y/N)
n
##########################################################Nearest Neighbour Query Completed ##########################################################################################



########################################################## Index Size Enquiry  ##############################################################################
Size of index created is: 445.0585021972656 MB
########################################################## Program Finished  ################################################################################
```

Nearest Neighbor Query Results and Index Size Reporting

Dataset 3 – coordinates_large.csv

```
############################################################### Build Tree ##################################################################

Starting with building the tree
Tree built successfully
Time taken to build the tree: 30811 milliseconds

############################################################### Tree Build Completed ########################################################################
Number of nodes 10000000
```

Build Tree timings and number of Nodes.

```
############################################################## Range Query ######################################################################
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
0
Enter lower bound of y
0
Enter lower bound of z
0
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 0.0, 0.0, 0.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 1236 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT1.csv having records equal to 10000000
Do you want to check again for a different range (Y/N)
y
Enter the lower bound of x, y and z coordinate values for finding range
Enter lower bound of x
550
Enter lower bound of y
550
Enter lower bound of z
550
Enter the upper bound of x, y and z coordinate values for finding range
Enter upper bound of x
1000
Enter upper bound of y
1000
Enter upper bound of z
1000
Starting with executing range query result between ( 550.0, 550.0, 550.0 ) and (1000.0, 1000.0, 1000.0 )
Range Result fetched in 231 milliseconds
Storing the result to output file
Range output written and saved successfully with name Q1-Output_KDT2.csv having records equal to 910943
Do you want to check again for a different range (Y/N)
n
##############################################################Range Query Completed ###########################################################################################
```

Range Query Results

```
######################################################## Nearest Neighbour Query ##########################################################################################
Enter the value of x, y and z coordinates of which you want to find the nearest neighbour for
Enter the value of x
56.5
Enter the value of y
86.04
Enter the value of z
71.51
Starting with executing NN query result for coordinates ( 56.5, 86.04, 71.51 )
NN result fetched in 1 milliseconds
NN of the entered coordinate are
[56.69, 85.23, 72.08]
Do you want to check NN again for a different coordinate (Y/N)
n
########################################################Nearest Neighbour Query Completed ##########################################################################################



######################################################## Index Size Enquiry  ##############################################################################
Size of index created is: 445.0747375488281 MB
######################################################## Program Finished  ##############################################################################
```

Nearest Neighbor Query Results and Index Size Reporting