k-d Tree for points in a plane

Vishwas Badarinath Sharma (ID : 108692460)
bitBucket code repo : https://bitbucket.org/csurfer/kdtree

1 Abstract

k-d Tree or k-dimensional tree is a space partitioning data structure for organizing points in a k-dimensional space. This data structure is quite handy when it comes to queries like range searches and nearest neighbor searches. As a class project for CSE-555 I would like to implement a data structure library for k-d trees in C++ focusing on 2 dimensional k-d trees. The purpose of the project would be to come up with an implementation of k-d tree data structure in C++ and compare its speed to the speed of the queries if processed in the Brute force way.

2 Supported Features

- Data structure called DataPoint which allows upto 2-d co-ordinate storage and supports functions like equals (Equality check), compare (Compare two points) and L2distance (L2 distance between two points) queries.

- Data structure called KDTree which allows upto 2-d co-ordinate storage and supports functions like display (Display KDTree), doesExists (Check existance of the given point in space), countRectange (Count the number of points that lie in the given rectangular boundary), reportRectangle (Report all the points that lie in the given rectangualr boundary), countCircle (Report all the points that lie in the given circular query), reportCircle (Report all the points that lie in the given circular query) queries.

- Implicitly supports countPoint (Count the number of points that lie on the given point), countLine(Count the number of points that lie on the given line) queries by specifying the inputs to earlier queries in a special format.

3 Implementation Details

KDTree is implemented as a class in C++ which can be instantiated in a simple way by the usage of its constructor which takes the kdTree dimensions and a vector of DataPoints as its parameters.

KDTree class uses two other classes called DataPoint (To store the data point) and Node (To store the node of the tree) internally. The current implementation follows the design model of input once and query many times i,e the input to KDTree has to be provided during the instantiation of the tree but the queries can be made any number of times.

4 Proof of Correctness

Proof of correctness is provided through implementation wherein it can be verified by matching the answer output by Brute Force method(bruteforce.cpp) and the KDTree method(kdtree.cpp). The only exception to this method of verification is report queries because in bruteforce approach the DataPoints are reported in the order of input and in the KDTree approach they are reported in the order they are found in the tree.

5 Algorithmic Analysis and Implementational Flaw

In the implementation of count and report Circle queries the implementation adds a factor of C to k where k are the reported DataPoints and hence the total complexity comes up to be $\mathrm{O(n^{1\text{-}1/d}} + \mathrm{Ck)}$.

This constant C is almost equal to 1 when the distribution of DataPoints is uniformly random. In a non-uniform distribution this constant can increase and lead to increased query time.

6 Time Analysis

Following are the details of the time it took to answer 2000 queries on 10000 data points by the brute-force and kd-tree approaches. The test input used was taken from /mediumtests folder. The timing is obtained from time command of linux shell in the following way

```
time ./a.out<test_input_file
```

The results obtained are as follows

Query	**Brute Force Approach**	**KDTree Approach**
Existence	0m0.450s	0m0.081s
Count Rectangle	0m0.683s	0m0.655s
Report Rectangle	0m6.107s	0m5.368s
Count Circle	**0m0.812s**	0m1.394s
Report Circle	**0m21.745s**	0m22.744s

Table 1

We can observe that in the emphasized time value Brute Force Approach beats the KDTree approach. The thing to note is

KDTree approach = Time to construct the tree + Time for querying

whereas in the case of BruteForce its just the query time. So how much time does it take to construct the tree in practice? Here are some values calculated exprimentally.

No of data points	**Tree build+Input read time**
1000	0m0.015s
10000	0m0.068s
100000	0m0.744s

From the above results we can clearly see that KDTree for a large number of data points can be constructed quite fast. However the time consumed to construct the tree is of significance when compared with total query time. This time should be considered as preprocessing time and hence it is clear that KDTree proves to be extremely fast when compared to the Brute Force approach and scales gracefully.

7 Code bundle contents

- /bigtests : Test cases which have 100000 Data points and 20000 query points.
- /mediumtests : Test cases which have 10000 Data points and 2000 query points.
- /smalltests : Test cases which have 100 Data points and 200 query points.
- /test_gen : Test case generater files.

- bruteforce.cpp : Bruteforce way to do existance check, circle count and report queries and rectangle count and report queries.
- kdtree.cpp : KDTree approach to the above problem.

8 How to run the code for checking ?

Comment or un-comment relevent parts of the kdtree.cpp main() function and save it. Then do

```
g++ kdtree.cpp
./a.out<test_input_file
```

9 Structure of test_inp_file

Very first line contains the dimensionality of the points. Second line contains the number of data points to follow(say n). The next n lines contain a data point in that a particular dimension. This is followed by n/5 number of query points which follow the format of one of these three :

- Existence Query : (Dimentionality of point) (Co-ordinates of Point)
- Rectangle Query : (Dimentionality of point) (Co-ordinates of Point 1) (Dimentionality of point) (Co-ordinates of Point 2)
- Circle Query : (Dimentionality of point) (Co-ordinates of Point) (Value of Radius of Circle)
- Point Query can be made by making Point 1 and Point 2 in Rectangle query the same and Line Query can be made by making Point 1 and Point 2 in Rectangle query co-linear.

10 Improvements possible

- Support for polygon queries.
- Removal of the constant factor in circle queries.

11 References

- Wikipedia article on KDTree.