Stony Brook University

FAR BEYOND

AMS 545 Optional Project: Implementation of KD Trees

Wenhan Gao

Stony Brook University

AMS 545 Optional Project

Implementation of KD Trees

Wenhan Gao

FAR BEYOND

Project Description

1. KD-trees for points in the plane will be implemented.
2. Timing experiments will be conducted. For various n, a set S of n random points in the unit square will be generated. The KD-tree for each S will be built. The time it takes to answer a query Q given by a box rectangle, $[0.45,\ 0.55] \times [0.45,\ 0.55]$, will be recorded.
3. Same experiments for "point in set" search. In other words, Q is now a "point" of the form $[0.45,\ 0.45] \times [0.55,\ 0.55]$.
4. KD-trees will also be used on an input set of axis-aligned rectangles so that those within a query box can be reported. This will be treated as a 4D range search problem.

Stony Brook University

Pseudo-code

Algorithm 2.1: buildKdTree(node, points, depth, k)

```
Result: node
Require: node, points, depth, k (dimension of the an input point)
if len(points) == 0 then
    return;
else if len(points) == 1 then
    node.val := points[0] ;
    return node;
else
    current dimension := depth mod k ;
    split_val := median of the values of points in current dimension;
    node.val = split_val;
    left_list, right_list := split points into two sets based on split_val and current dimension;
    node.left = Node();
    node.right = Node();
    node.left = buildKdTree(node.left, left_list, depth+1, k);
    node.right = buildKdTree(node.right, right_list, depth+1, k);
end
```

Algorithm 2.2: queryRecursion(node, rectangle, output, depth, k)

```
Result: resulting points stored in an array
Require: node, rectangle, output, depth=0, k
if node.val == None then
    return;
if node is a leaf node and the point node.val is in the rectangle then
    append node.val to output array ;
    return;
current dimension = depth mod k;
if the left region of current node intersect the rectangle in the current dimension then
    queryRecursion(node.left, rectangle, output, depth+1, k);
if the right region of current node intersect the rectangle in the current dimension then
    queryRecursion(node.right, rectangle, output, depth+1, k);
```

In a nutshell, the construction of KD trees is done by recursion. Each time, the list of input “points” is split in half by the median value in “current dimension” to construct the left and right subtrees.

The query is done by recursion as well. If the right or left region intersect the query box, we descent a path down the right or left child, possibly both, until a leaf node is reached, and we test if leaf node is in the query box.

FAR BEYOND

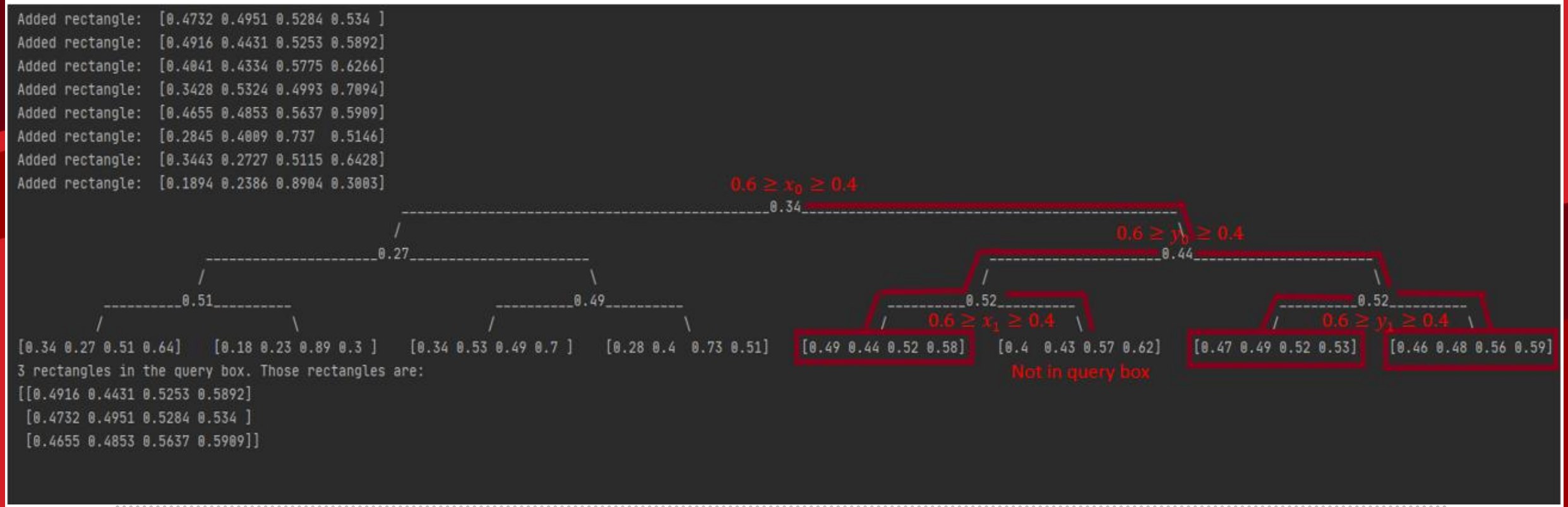
Stony Brook University

Program Demo

Stony Brook University

Example of a KD Tree in 4D

Set of input rectangles in the form of $[x_0, y_0, x_1, y_1]$. Query Box:$[0.4,\ 0.6] \times [0.4,\ 0.6]$, which can be viewed as $[0.4,\ 0.6] \times [0.4,\ 0.6] \times [0.4,\ 0.6] \times [0.4,\ 0.6]$ in 4D.

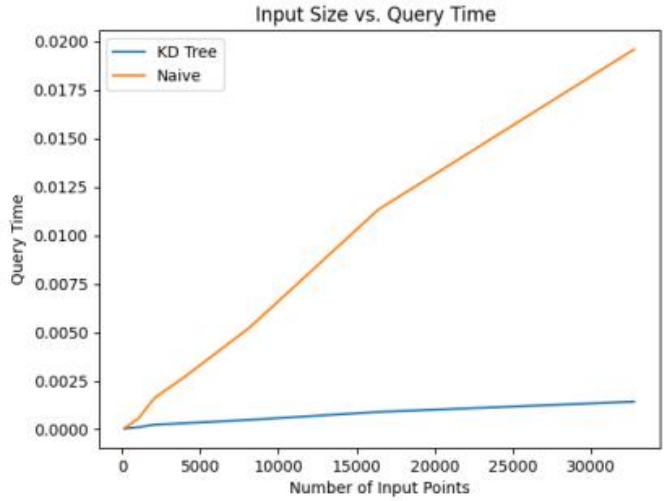


FAR BEYOND

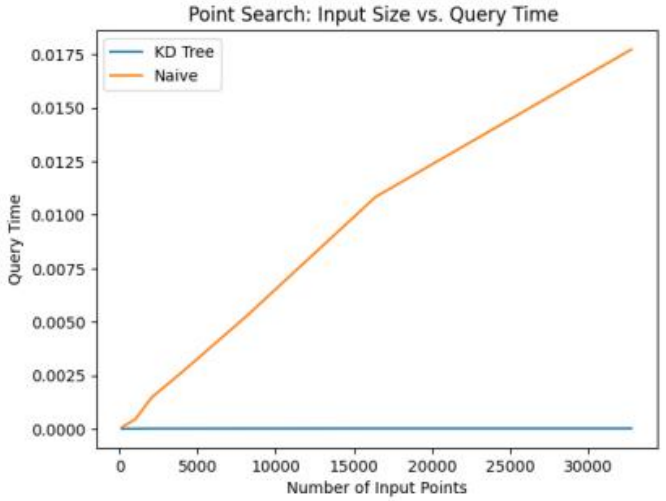
Stony Brook University

Timing Results

Figure 5: Query Time Comparison



(a) Range Search



(b)Point Search

For reference, the query time for KD-tree is $O(\sqrt{n}+k)$, where k is the number of points in the query rectangle. Since the query rectange is a fixed 0.1 by 0.1 box, and the input points are random, the expected size of k is $0.1n$. Therefore, when n is large, k will dominate $\sqrt{n}$ in expectation; KD tree query time is expected $O(\frac{1}{10}n)$ when n is large. The naive method query runs in $O(n)$ time.

However, if we consider the point search problem, the KD-tree querry runs in $O(\sqrt{n})$ time whereas the naive method still runs in $O(n)$ time.

FAR BEYOND

Stony Brook University

Additional Timing Results

Appendix A Timing Results

Table 1: Range Search: Average Query Time

	KD tree	Naive Method
$N = 2^7$	3.316×10^{-5}	6.532×10^{-5}
$N = 2^8$	5.747×10^{-5}	1.410×10^{-4}
$N = 2^9$	8.174×10^{-5}	2.759×10^{-4}
$N = 2^{10}$	1.165×10^{-4}	5.560×10^{-4}
$N = 2^{11}$	2.456×10^{-4}	1.608×10^{-3}
$N = 2^{12}$	3.256×10^{-4}	2.772×10^{-3}
$N = 2^{13}$	4.959×10^{-4}	5.277×10^{-3}
$N = 2^{14}$	9.009×10^{-4}	1.132×10^{-2}
$N = 2^{15}$	1.430×10^{-3}	1.958×10^{-2}

Time to build the KD Tree is (in ascending order for input sizes from 2^7 to 2^{15}):

$$[0.002366\ 0.007223\ 0.01139\ 0.02325\ 0.07923\ 0.1508\ 0.3302\ 0.6345\ 1.216]$$

Table 2: Point Search: Average Query Time

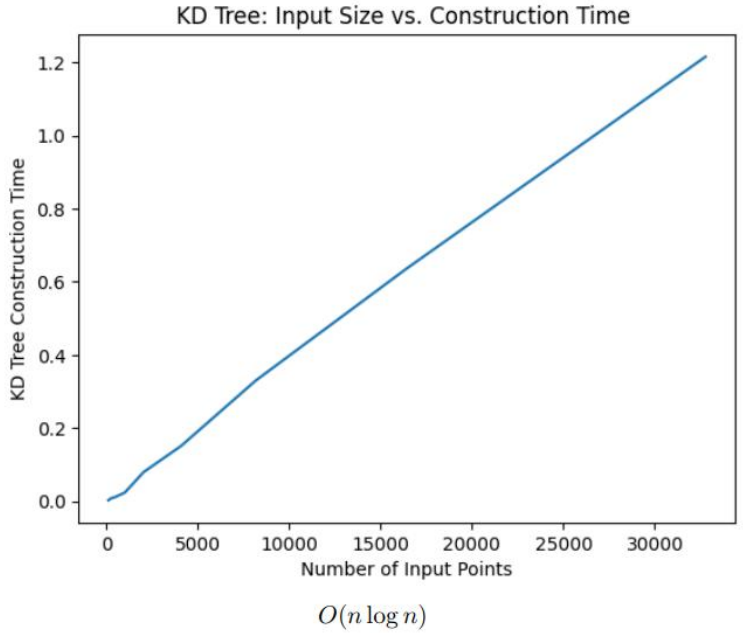
	KD tree	Naive Method
$N = 2^7$	8.66×10^{-6}	5.81×10^{-5}
$N = 2^8$	1.32×10^{-5}	1.34×10^{-4}
$N = 2^9$	1.23×10^{-5}	2.42×10^{-4}
$N = 2^{10}$	1.30×10^{-5}	4.68×10^{-4}
$N = 2^{11}$	1.94×10^{-5}	1.47×10^{-3}
$N = 2^{12}$	2.00×10^{-5}	2.72×10^{-3}
$N = 2^{13}$	2.19×10^{-5}	5.32×10^{-3}
$N = 2^{14}$	2.60×10^{-5}	1.08×10^{-2}
$N = 2^{15}$	2.54×10^{-5}	1.77×10^{-2}

The KD Tree structure is much faster than the naïve method!

Stony Brook University

Additional Timing Results

Figure 8: KD Tree: Input Size vs. Construction Time



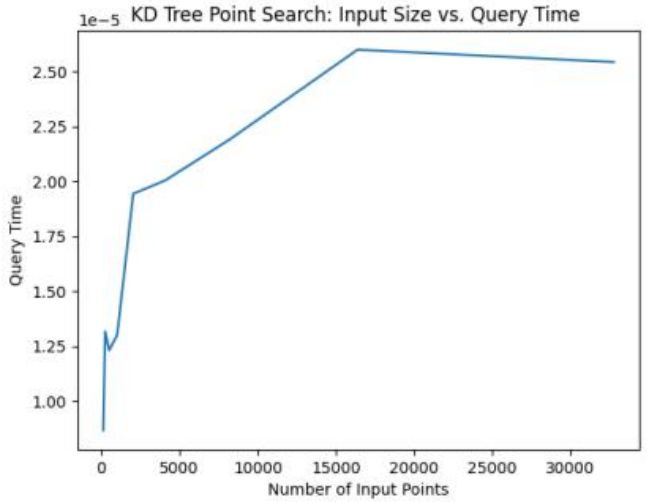
KD Tree construction is $O(n \log n)$, almost linear!

FAR BEYOND

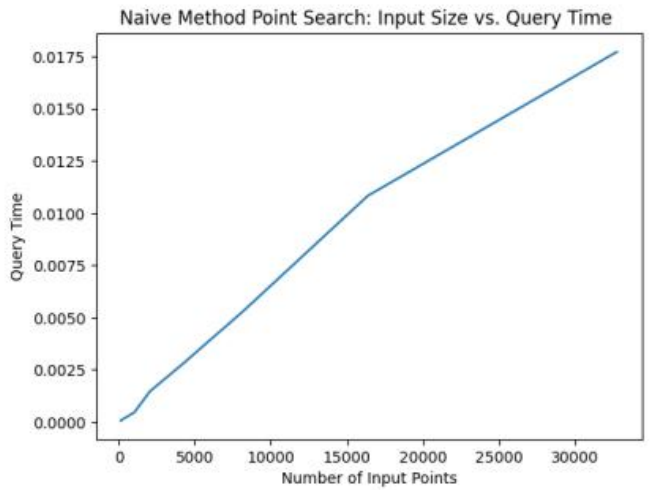
Stony Brook University

Additional Timing Results

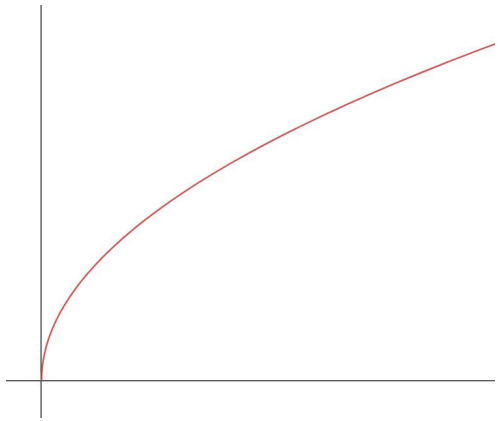
Figure 7: Plot: Point Search Query Time



(a) KD Tree; $O(\sqrt{n})$



(b)Naive Method; $O(n)$



Reference: plot of $\sqrt{x}$

The KD Tree structure is much faster than the naïve method!

FAR BEYOND

Thank you!