The kd-Tree

[Notes for the Training Camp]

Yufei Tao

ITEE
University of Queensland

All our structures so far are "one-dimensional", in the sense that each data element is an integer. In this talk, we will see our first two-dimensional data structure—named the kd-tree—for the classic problem of range reporting.

Range Reporting

Let $\mathbb{R}$ denote the set of real values. Let P be a set of n points in $\mathbb{R}^2$ (namely, each point has the form (x, y) where x and y are real values).

Given an axis-parallel rectangle q (namely q has the form $[x_1, x_2] \times [y_1, y_2]$), a range reporting query returns all the points of P that are covered by q.

Our objective is to store P in a data structure, so that we can answer all queries efficiently.

Think: How would you solve the range reporting problem in one-dimensional space?

The kd-tree that we will learn next solves the problem with $O(n)$ space and $O(\sqrt{n} + k)$ query time, where k is the number of points reported.

kd-Tree

We will describe the structure in a recursive manner.

Base Case: If $n = 0$, the tree is empty. If $n = 1$, then the tree has a single node which stores the only point in P.

General Case: $n \geq 2$. Find a vertical line ℓ_x that divides P as evenly as possible. Let P_{left} be the set of points of P on the left of ℓ_x, and similar, P_{right} the set of points on the right.

Create a root node r, and store ℓ_x at r.

If P_{left} has only 1 point, create a leaf node storing this point as the left child of r. Similarly, if P_{right} has only 1 point, create a leaf node storing this point as the right child of r.

Next, we consider that P_{left} and P_{right} have at least 2 points.

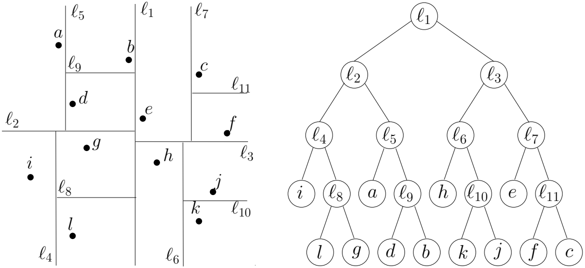
kd-Tree

General Case (cont.): Find a horizontal line ℓ_{y1} that divides P_{left} as evenly as possible into two sets P_1, P_2. Similarly, find a horizontal line ℓ_{y2} that divides P_{right} as evenly as possible into two sets P_3, P_4.

Create a node u_1 as the left child of r, and u_2 as the right child of r. Store l_{y1} at u_1, and l_{y2} at u_2. Then comes the recursive construction:

- Create a kd-tree on P_1, and make its root the left child of u_1.
- Create a kd-tree on P_2, and make its root the right child of u_1.
- Create a kd-tree on P_3, and make its root the left child of u_2.
- Create a kd-tree on P_4, and make its root the right child of u_2.

Example



Observe that every node corresponds to a rectangle in the data space. For example, the root corresponds to the entire data space. Its right child corresponds to the part of the data space on the right of ℓ_1. The node l_6 corresponds to the part that is to the right of ℓ_1, and below ℓ_3.

Query

Given a node u, let $R(u)$ be the rectangle in the data space that u corresponds to.

Given a query with rectangle q, we answer it as follows:

- Simply visit all the nodes u whose $R(u)$ intersects q.
- At a leaf z, if the point p at z falls into q, report p.

The kd-tree clearly occupies $O(n)$ space—noticing that every internal node of the kd-tree must have 2 children (why?), and that the tree has n leaves.

It is easy to construct the tree in $O(n \log n)$ time (how?).

Next, we will prove that the query algorithm has running time $O(\sqrt{n} + k)$.

Analysis

Given a vertical/horizontal line ℓ, we say that ℓ intersects a node u if ℓ intersects $R(u)$.

The following is an important lemma behind the query efficiency:

Lemma: Any vertical line intersects $O(\sqrt{n})$ nodes.

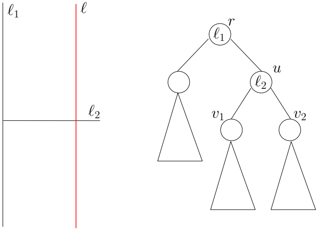
Analysis

Proof: Denote by $f(n)$ the largest number of nodes in a kd-tree of n points that can be intersected by any vertical line. Let us analyze what these nodes are.

First, the root r itself, whose rectangle is the whole universe, obviously intersects ℓ. Let us assume, without loss of generality, that ℓ is to the right of the split line at r. Let u be the right child of r. Clearly, ℓ also intersects $R(u)$.

The next slide shows a figure about this.

Analysis



How about the nodes in the subtrees rooted at v_1 and v_2? How many of them intersect ℓ? Observe that, each subtree is a kd-tree on at most $n/4$ nodes! So the answer is at most $f(n/4)$ per subtree!

Analysis

Therefore, we have obtained an recurrence about $f(n)$:

$$f(n) \leq 2 + 2f(n/4).$$

The terminating condition is $f(1) = 1$.

Solving the recurrence gives $f(n) = O(\sqrt{n})$ $\square$.

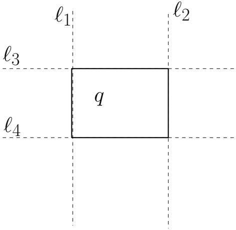
We can also prove that any horizontal line intersects the rectangles of $O(\sqrt{n})$ noes.

Analysis

We are now ready for our grand theorem:

Theorem: The kd-tree answers a query in $O(\sqrt{n} + k)$ time.

Proof: Let $\ell_1, \ell_2, \ell_3, \ell_4$ be the vertical or horizontal lines defining the boundary of the query rectangle q, as shown below:



Analysis

Recall that every node u visited by the query must have its $R(u)$ intersecting q. We divide such nodes u into two categories:

- **Category 1:** $R(u)$ intersects at least one edge of q.
- **Category 2:** $R(u)$ falls completely inside q.

How many nodes are there in the first category? The answer is at most $4\sqrt{n}$—noticing that every node u of this category must have $R(u)$ intersecting one of ℓ_1, ℓ_2, ℓ_3 and ℓ_4!

So it remains to bound the number of nodes in Category 2. We will show that there are only $O(k)$ of them. This will complete the proof that the query time is $O(\sqrt{n} + k)$.

Analysis

Let v be a node in Category 2. Call v a top node if its parent is not in Category 2. Observe that:

- The subtrees of two top nodes must be disjoint (i.e., no top node can be a proper descendant of another).
- Every node in Category 2 must be in the subtree of a top node.
- Every leaf descendant of a top node must store a point falling in q.

Since the number of internal nodes in each subtree equals the number of leaves in that subtree minus 1, the above facts imply that Category 2 has $O(k)$ nodes. □

We have seen range reporting in 1d space, and 2d space. The problem definition extends to any d-dimensional space, where d is an integer.

The kd-tree, too, can be extended! In d-dimensional space (where $d = O(1)$ and $d \geq 2$), it still uses $O(n)$ space, and answers a query in $O(n^{1-1/d} + k)$ time.

How?