Range Tree

Yunsen Liang

Taishan College

Table of Contents

Introduction

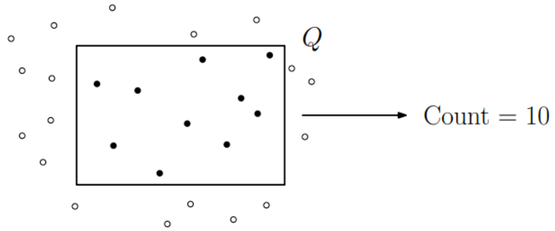


图:

- Let S be a set of points in $\mathbb{R}^2$. Given an axis-parallel rectangle Q, a range query retruns all the points of S that are covered by Q, namely, $S \cap q$
- The definition can be extended to any dimensionality in a straightforward manner

Introduction

Essentially, this is a two-dimensional partial ordering problem

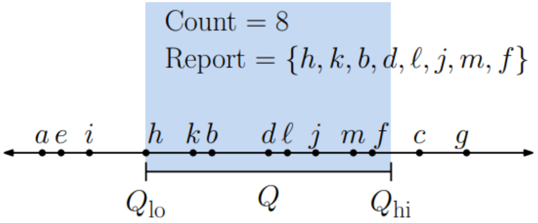
- 2-d prefix sum
- weighted segment tree、binary indexed tree
- k-d tree
- R tree
- ...

But today I want to introduce range trees to provide a new way to solve this problem

Introduction

- The 2-dimensional range query is essentially composed of two 1-dimensional sub-queries, one on the x-coordinate of the points and one on the y-coordinate
- In other words, we can only care about one dimension first

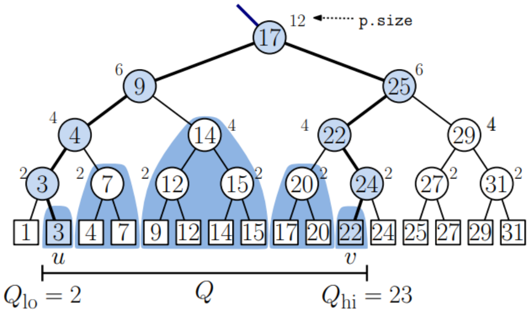
1-dimensional range tree



$\text{Count} = 8$
$\text{Report} = \{h, k, b, d, \ell, j, m, f\}$
a e i h k b d ℓ j m f c g
Q_{lo}
Q
Q_{hi}

Question: How to efficiently find all the points between the two boundary points?

1-dimensional range tree



- Node
  - size, the external nodes in p's subtree
  - left, point to the left subtree
  - right, point to the right subtree
  - isLeaf, we call the leaves as external nodes, the remaining as inner nodes
- e.g. we illustrate for the interval [2, 23]

1-d range query algorithm

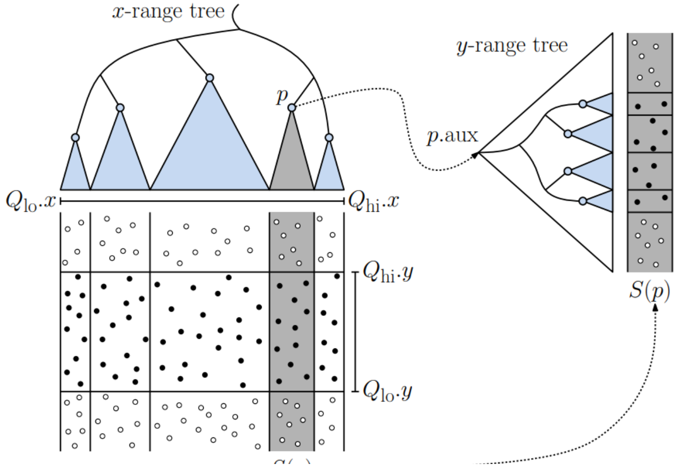
Algorithm 1: 1DRangeQuery(T, $[x_1 : x_2]$)

```
if p is external node then
    if p in range $[x_1, x_2]$ then
        return 1
    else
        return 0
    end
else if $[x_1, x_2]$ contains p then
    return p.size
else if p disjoint from $[x_1, x_2]$ then
    return 0
return 1DRangeQuery(T(left), $x_1, x_2$) + 1DRangeQuery(T(right), $x_1, x_2$)
```

theorem

1-dimensional range counting queries can be answered in O(log n) time and range reporting queries can be answered in O(k + log n) time, where k is the number of values reported.

2-dimensional range tree



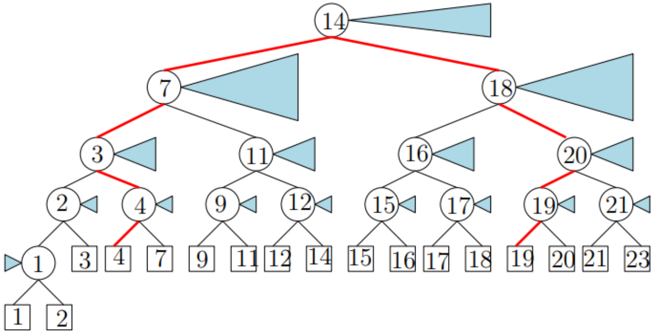
- We first build a range tree based on the x-coordinate, and then for each point, we build another range tree based on the y-coordinate.
- Each node has an additional pointer "aux", pointing to the range tree built based on the y-coordinate

Construction algorithm

- $T(n) = 2 \cdot T(n/2) + O(n)$
- The construction algorithm takes $O(n \log n)$ time

2-d range query algorithm

- Query $[4:19] \times [y_1 : y_2]$



- Similar to the approach in one dimension, start by finding v_{split}
- For points along the path starting from the split that meet the x criteria, query the y-coordinate

2-d range query algorithm

Algorithm 2: 2DRangeQuery(T, $[x_1 : x_2] \times [y_1 : y_2]$)

```
$v_{split} \leftarrow$ FindSplitNode(T, $x_1, x_2$);
if $v_{split}$ is a leaf then
    Check if the point in $v_{split}$ must be reported.
else
    $v \leftarrow lc(v_{split})$
    while $v$ is not a leaf do
        if $x_1 \leq value(v)$ then
            1DRangeQuery($T_y(rc(v)), [y_1 : y_2]$)
            $v \leftarrow lc(v)$
        else
            $v \leftarrow rc(v)$
        end
    end
    Check if the point in v must reported.
    Similarly, follow the patch from rc($v_{split}$) to $x_2$
end
```

2-d range query efficiency

- We search in $O(\log n)$ associated structures to perform a 1D range query
- The query time is $O(\log^2 n + k)$, where k is the size of the output

theorem

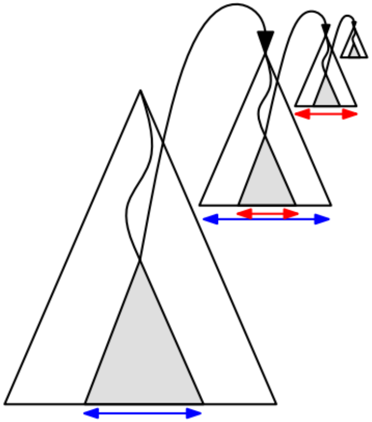
A set of n points in the plane can be processed in $O(n \log n)$ time into a data structure of $O(n \log n)$ size so that any 2D range query can be answered in $O(\log^2 n + k)$ time, where k is the number of answers reported.

Recall that a k-d tree has $O(n)$ size and answers queries in $O(\sqrt{n} + k)$ time

My own understanding

- It feels like a range tree is very similar to a segment tree
- The range tree is divided by taking the median
- Therefore, some voices on the Internet say that a two-dimensional segment tree is also a two-dimensional range tree

Higher dimensional range trees



A d-dimensional range tree has a main tree which is a one-dimensional balanced binary tree on the first coordinate, where every node has a pointer to an associated structure that is a (d-1)-dimensional range tree on the other coordinates.

Storage

$$S_1(n) = O(n) \quad \textit{for all } n$$

$$S_d(1) = O(1) \quad \textit{for all } d$$

$$S_d(n) \leq 2 \times S_d(n/2) + S_{d-1}(n) \quad \textit{for } d \geq 2$$

- This solves to $S_d(n) = O(n \log^d n)$

Query time

- The number of grey nodes $G_d(n)$ satisfies:

$$G_1(n) = O(\log n) \quad \textit{for all } n$$

$$G_d(1) = O(1) \quad \textit{for all } d$$

$$G_d(n) \leq 2 \times \log n \times G_{d-1}(n) \quad \textit{for } d \geq 2$$

- This solves to $G_d(n)$ =O(n $\log^d n$)
- In fact, we can also use induction to prove this

Result

theorem

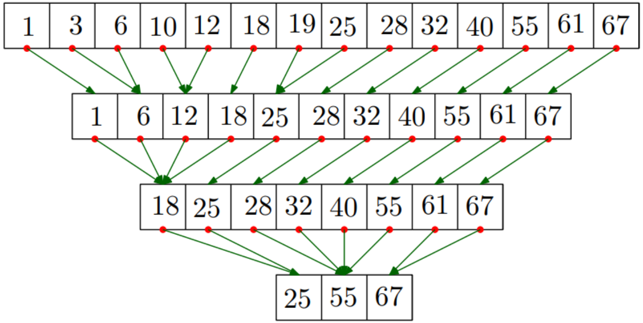
A set of n points in a d-dimensional space can be processed in $O(n \log^{d-1} n)$ time into a data structure of $O(n \log^{d-1} n)$ size so that any k-D range query can be answered in $O(\log^d n + k)$ time, where k is the number of answers reported.

Recall that a kd-tree has $O(n)$ size and answers queries in $O(n^{1-1/d} + k)$ time

Fractional cascading

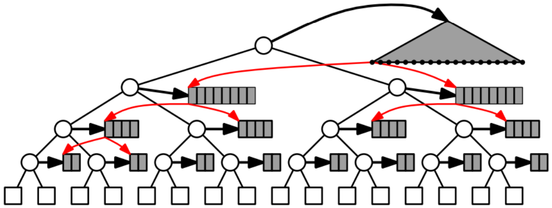
- We can improve the query time of a 2D range tree from $O(\log^2 n)$ to $O(\log n)$ by a technique called **fractional cascading**
- The idea illustrated best by a diffetent query problem:
  Suppose that we have a collection of sets $S_1 ... S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure taht can report for a query number x, the smaller value greater than equals to x in all sets $S_1 ... S_m$

Fractional cascading



- Each number in the upper level is connected to the first number in the lower level that is greater than or equal to it.
- e.g. We want to find the first number that greater than 33.

Fractional cascading



- Now we do "the same" on the associated structures of a 2-dimensional range tree
- Note that om every associated structure, we search with the same values y_1 and y_2
- Replace all associated structures(y trees) with sorted lists
- For every list element (and leaf of the associated structure of the root), store two pointers to the appropriate list elements in the lists of the left child and of the right child

Result

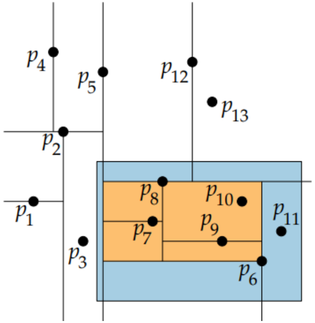
theorem

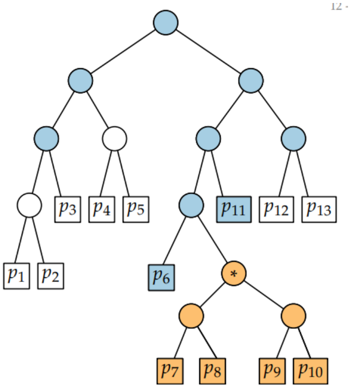
A set of n points in in d-dimensional space can be preprocessed in O($nlog^d n$) time into a data structure of O($nlog^{d-1} n$) size so that any k-D range query can be answered in O($log^{d-1} n + k$) time,where k is the number of answers reported.

Degenerate cases

- Both for kd-trees and for range trees we have to take care of multiple points with the same x- or y-coordinate
- We define a composite number as a pair of reals, denoted by $(a|b)$
- We let $(a|b) < (c|d)$ iff $a < c$ or ($a = c$ and $b < d$); this defines a total order on composite numbers
- The point $p = (p_x, p_y)$ becomes $((p_x|p_y, (p_y|p_x)))$. Then no two points have the same first or second coordinate
- The median x-coordinate or y-coordinate is a composite number

Complexity analysis





- The time complexity is primarily based on the number of intersections with the rectangular boundaries
- let the number be denoted by $Q(n)$

Complexity analysis

- $Q(n) = \begin{cases} O(1) & \textit{if} \quad n = 1 \\ O(1) + 2Q(n/4) & \textit{else.} \end{cases}$
  $Q(n) = O(\sqrt{n})$
- Similarly, we can generalize this to k dimensions
- $Q(n) = \begin{cases} O(1) & \textit{if} \quad n = 1 \\ O(1) + 2^{k-1}Q(n/2^k) & \textit{else.} \end{cases}$
  $Q(n) = O(n^{1-1/k})$

derivation process

- Use the conclusions of the Master Theorem
- Use knowledge of sequences

Comparison

Data Structure	Complexity		
	Construction	Query	Space
KD Tree (2D)	$O(n\log n)$	$O(\sqrt{n})$	$O(n)$
KD Tree (kD)	$O(n\log n)$	$O(n^{1-1/k})$	$O(n)$
Range Tree (2D)	$O(n\log n)$	$O(\log^2 n)$	$O(n\log n)$
Range Tree (kD)	$O(n\log^{k-1} n)$	$O(\log^k n)$	$O(n\log^{k-1} n)$

Comparison

- Similarities:
  - They can both address high-dimensional problems
  - They both employ a method of analyzing one dimension at a time
  - They require maintaining balance to ensure query efficiency
  - Typically, they are static and do not involve inserting or deleting data.
- Differences:
  - In contrast, the range tree appears to be more specialized in addressing range queries
  - K-D Tree is commonly used for nearest neighbor queries
  - K-D Tree: query path alternates between x-coordinate and y-coordinate
  - Range tree: first-level tree for x-coordinates and many second-level trees for y-coordinates

Application

- Database query
  Each type of information about an object can be abstracted into a dimension.
- Computational geometry
  - nearest neighbor query
  - interval intersection
  - quickly detect whether there is an intersection between two three-dimensional objects