CSE520 Computational Geometry
Lecture 8
Orthogonal Range Searching

Antoine Vigneron

Ulsan National Institute of Science and Technology

June 15, 2020

Outline

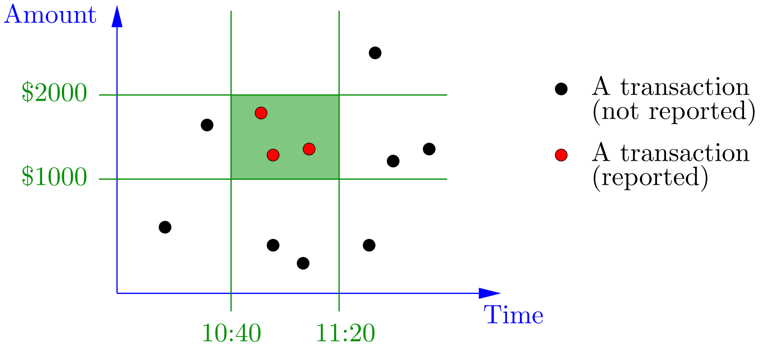
- In this lecture, we consider the problem of querying a set of points.
- Solution in one dimension.
- Data structure in $\mathbb{R}^2$: Range trees.
- Extension to higher dimensions.
- $\log n$ factor improvement

Reference: Textbook Chapter 5.

Example

- A database records financial transactions.
- Query: Find all the transactions such that
  - The amount is between \$ 1000 and \$ 2000,
  - and it happened between 10:40 am and 11:20 am.

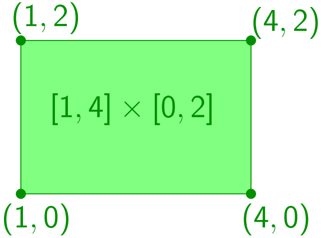
Geometric interpretation:



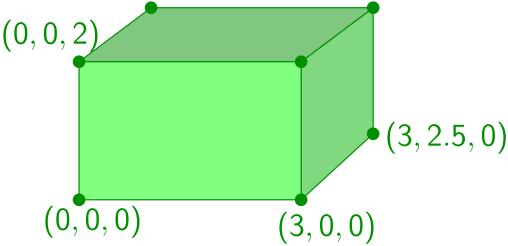
Query Problems

- Let n denote the total number of transactions in the database.
- We will show how to build in $O(n \log n)$ time a data structure of size $O(n \log n)$ that allows to answer this type of queries in $O(k + \log n)$ time, where k is the size of the output (the number of transactions that are reported).
- The data structure is built only once, then queries should be answered quickly.
- We say that:
  - The *space usage* is $O(n \log n)$,
  - the *preprocessing time* is $O(n \log n)$,
  - and the *query time* is $O(k + \log n)$,

Boxes



A 2D-box is a an axis-parallel rectangle.



The 3D-box $[0, 3] \times [0, 2.5] \times [0, 2]$

Definition (Box)

A box in $\mathbb{R}^d$ is the set $[a_1, b_1] \times \cdots \times [a_d, b_d]$ for some $a_1, \ldots, a_d, b_1, \ldots, b_d \in \mathbb{R}$.

- Algorithmic problems involving boxes are often much easier than problems involving general polytopes.

Problem Statement

Problem (Orthogonal range searching)

Let P be a set of n points in $\mathbb{R}^d$. Preprocess P so as to answer the following queries efficiently:

- INPUT: $a_1, \dots, a_d, b_1, \dots, b_d$.
- OUTPUT: $P \cap ([a_1, b_1] \times \cdots \times [a_d, b_d])$.

- We denote $k = |P \cap ([a_1, b_1] \times \cdots \times [a_d, b_d])|$.
- In other words, k is the size of the output.

- We assume that $d = O(1)$. (*fixed* dimension.)

One Dimensional Case ($d = 1$)

- $P \subset \mathbb{R}$.
- Queries: Find all the numbers in P that are between a and b.
- Data structure:
  - A balanced binary search tree (BBST), for instance a red-black tree.
  - Preprocessing time: $\Theta(n \log n)$ time to build a BBST.
  - Space usage: $\Theta(n)$.
- Query time: $\Theta(k + \log n)$ time. (See next slides.)
- It can also be done with an array.
  - It is easier with an array when $d = 1$.
  - But the BBST approach generalizes to $d > 1$.

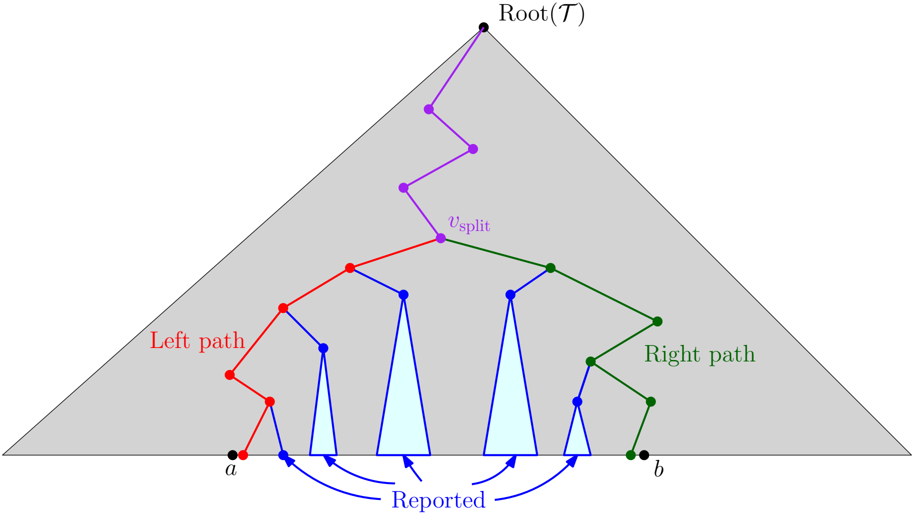
Answering a Query

One-dimensional range queries

Algorithm *Report*$(\mathcal{T}, a, b)$
Input: a BBST $\mathcal{T}$ storing P, an interval $[a, b]$
Output: $P \cap [a, b]$

if $\mathcal{T} = NULL$
    then return
x $\leftarrow$value stored at the root of $\mathcal{T}$
if $a \leqslant x$
    then $Report(\mathcal{T}.left, a, b)$
if $a \leqslant x \leqslant b$
    then output x
if $x \leqslant b$
    then $Report(\mathcal{T}.right, a, b)$

Analysis



Analysis

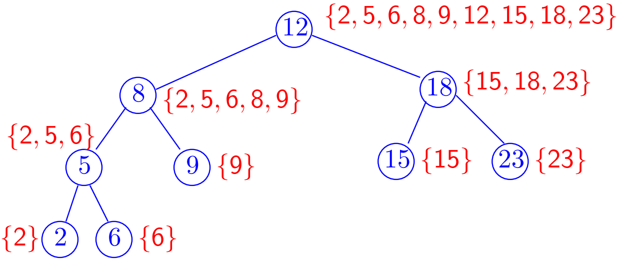
- The search paths to a and b consist of:
  - A common subpath from the root to a vertex v_{split}.
  - A subpath from v_{split} to a and b, respectively.
- When answering a query, we visit the left path, the right path, v_{split}, and the subtrees in between.
- Path lengths are $O(\log n)$:
  - path from root to v_{split},
  - left path,
  - right path.
- Sum of the sizes of blue subtrees $\leqslant k$.
- Query time: $O(k + \log n)$.

Planar Case (d=2) using Range Trees

- INPUT: a set P of n points in $\mathbb{R}^2$.
- QUERY: Given (a_1, a_2, b_1, b_2), find the points $(x, y) \in P$ such that $x \in [a_1, b_1]$ and $y \in [a_2, b_2]$.
- Results presented in this section:
  - $\Theta(n \log n)$ preprocessing time,
  - $\Theta(n \log n)$ space usage,
  - $\Theta(k + \log^2 n)$ query time.
- Query time will be improved to $O(k + \log n)$ in the last section.

Canonical Sets

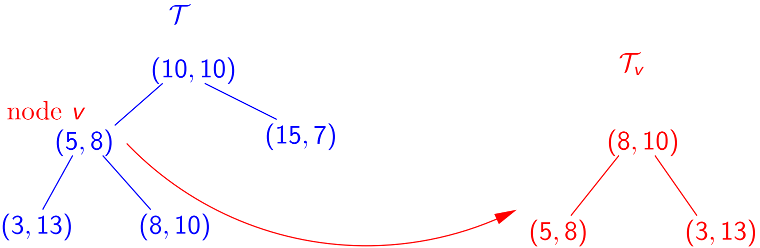
- We first store P in a BBST $\mathcal{T}$ with the x-coordinates as keys.
- Each node v of $\mathcal{T}$ is associated with a *canonical set* C_v, which is the set of all the points in P that are stored in the subtree rooted at v.



Example of canonical sets

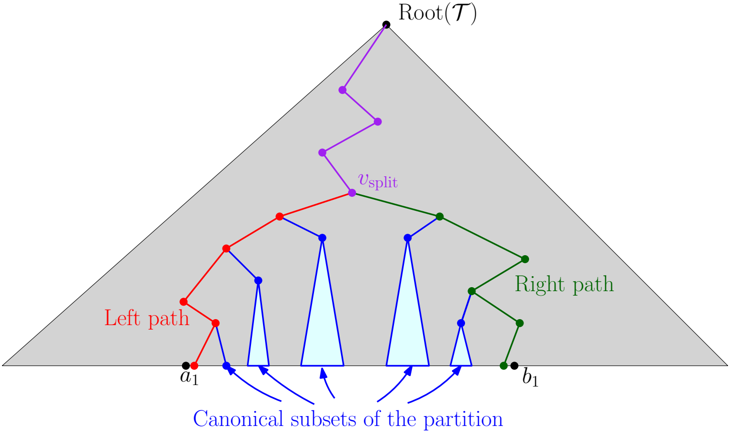
Range Trees in $\mathbb{R}^2$

- The canonical set of each node v of $\mathcal{T}$ is stored in a BBST $\mathcal{T}_v$ with the y-coordinates as keys.



Querying a Range Tree

- Let $P' = P \cap ([a_1, b_1] \times \mathbb{R})$.
- Let P'' be the set of points on the right path and the left path.
- We partition $P' \setminus P''$ into canonical subsets.
  - $P' = P'' \cup C_1 \cup C_2 \cup \dots C_c$.



Partitioning P'

- Find the left path and the right path, which gives P''.
- Pick each canonical set stored in a node that is not in the left path, but is the right child of a node of the left path.
- Pick each canonical set that is stored in a node that is not on the right path, but is the left child of a node of the right path.
- Thus we obtain canonical sets $C_1, \dots, C_c$.
- It takes $O(\log n)$ time (height of the BBST).
- We have picked $c = O(\log n)$ canonical sets C_i.

Querying a Range Tree

- $\forall p \in P''$ check if $p \in [a_1, b_1] \times [a_2, b_2]$, and report it if it the case.
- $\forall i$ perform a one dimensional range searching query in C_i with the interval $[a_2, b_2]$ (using the appropriate tree $\mathcal{T}_{v_i}$).
- The union of all these results gives $P \cap ([a_1, b_1] \times [a_2, b_2])$.
- Let k_i be the number of points reported in C_i.
- Analysis:

$$\sum_{i=1}^{c} k_i + \log n \leqslant k + c \log n$$
$$= k + O(\log^2 n)$$

so the query time is $O(k + \log^2 n)$.

Space Usage

- For each node v, let C_v denote the canonical set at node v.
- A point p belongs to all the canonical sets in the path from the vertex of $\mathcal{T}$ that stores p to the root (and only these canonical sets).
- Hence it belongs to $O(\log n)$ canonical sets.
- So
$$\sum_{v \in \mathcal{T}} |C_v| = O(n \log n).$$
- Space usage is $O(n \log n)$.
- More precisely, it is $\Theta(n \log n)$.
    - Why?

Preprocessing Time

- T_v can be build in $O(|C_v| \log |C_v|)$ time.
- Hence the range tree can be build in time proportional to

$$\begin{aligned}\sum_v |C_v| \log |C_v| &= O(\log n). \sum_v |C_v| \\ &= O(n \log^2 n).\end{aligned}$$

- We can do better:
  - Compute the T_v's from leaves to root.
  - Obtain each T_v by merging two sorted sequences (from its children).
    - It takes $O(|C_v|)$ time.
  - Overall, we can build the range tree in time proportional to

$$\sum_v |C_v| = \Theta(n \log n).$$

Range Trees in Higher Dimension

Idea:

- We want to perform range searching in $\mathbb{R}^d$.
- We still build $\mathcal{T}$ with respect to the x_1-coordinate.
- For each canonical set of $\mathcal{T}$ we build a $(d-1)$-dimensional range searching data structure using coordinates $(x_2, x_3, \dots x_d)$.
- To answer a d-dimensional query:
  - Find the canonical sets of $\mathcal{T}$ associated with $[a_1, b_1]$.
  - Make a $(d-1)$-dimensional query on each canonical set recursively, using $[a_2, b_2] \times [a_3, b_3] \times \dots [a_d, b_d]$.
  - Check the points on the left and right path.

Analysis

- We assume $d > 1$ and $d = O(1)$.
- Query time: $O(k + \log^d n)$.
- Space usage: $\Theta(n \log^{d-1} n)$.
- Preprocessing time: $\Theta(n \log^{d-1} n)$.
- Proof:
  - By induction on d.

Improved Range Trees

Motivation:

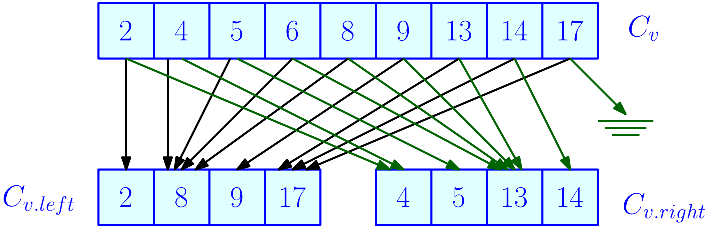
- In $\mathbb{R}^2$, range trees have query time $\Theta(k + \log^2 n)$.
- Can we do better?
- Yes, in this section we obtain a $\Theta(k + \log n)$ query time.

Improved Range Trees

Idea:

- When processing a query (a_1, a_2, b_1, b_2), we search several trees T_v, always with the same key b_1 or b_2.
- For each such tree we spend $O(\log n)$ time.
- $C_{v.left}$ and $C_{v.right}$ are subsets of C_v.
- We will keep pointers from nodes of $\mathcal{T}_v$ to nodes of $\mathcal{T}_{v.left}$ and $\mathcal{T}_{v.right}$ that hold the same key, or the next key.
- After performing a search in $\mathcal{T}_v$, it will allow to perform a search in $\mathcal{T}_{v.left}$ and $\mathcal{T}_{v.right}$ in $O(1)$ time.

Data Structure



- We first perform a search in C_{root}, which takes $O(\log n)$ time.
- While searching $\mathcal{T}$, we follow these pointers from the parent of each node we visit.
- So during the search, we know the location of a_2 and b_2 in the canonical set of the current node.
- So a search in each C_i is performed in $O(1)$ time.

Consequences

- This technique is known as fractional cascading.
- By induction, it also improves by a factor $O(\log n)$ the results in $d > 2$.

Theorem

Range trees with fractional cascading allow to answer orthogonal range queries in dimension $d \geqslant 2$ within the following time and space bounds:

- *Query time* $\Theta(k + \log^{d-1} n)$,
- *Space usage* $\Theta(n \log^{d-1} n)$,
- *Preprocessing time* $\Theta(n \log^{d-1} n)$.

Concluding Remarks

- Range trees:
  - Simple,
  - nearly optimal.
- Spatial databases mainly use R-trees.
  - Not covered in CSE520.
  - Good in practice with usual datasets.
  - But no performance guarantee: no good worst case bound on the query time.