Lecture 7
Orthogonal range searching

Benjamin Sach

Orthogonal range searching

A d-dimensional range searching data structure stores n distinct \((x, y)\)-pairs and supports:

- the lookup\((x_1, x_2, y_1, y_2)\) operation

which returns every point in the rectangle \([x_1 : x_2] \times [y_1 : y_2]\)

i.e. every \((x, y)\) with \(x_1 \leq x \leq x_2\) and \(y_1 \leq y \leq y_2\).

A classic database query

"find all employees aged between 21 and 48 with salaries between 23,000 and 36,000 GBP"

Starting simple... 1D range searching

- build a sorted array containing the \(x\)-coordinates
- in \(O(n \log n)\) preprocessing (prep) time and \(O(n)\) space

- to perform lookup\((x_1, x_2)\)...
- find the successor of \(x_1\) by binary search and 'walk' right

- lookups take \(O(\log n + k)\) time (\(k\) is the number of points reported)

Starting simple... 1D range searching

Alternatively we could build a balanced tree...

We can store the tree in \(O(n)\) space (it has one node per point)

It has \(O(\log n)\) depth and can be built in \(O(n \log n)\) time (\(O(n)\) after sorting)

Starting simple... 1D range searching

- how do we do a lookup?

- find the successor of \(x_1\) in \(O(\log n)\) time

and the predecessor of \(x_2\) in \(O(\log n)\) time

- which points in the tree should we include?
Starting simple... 1D range searching

look at any node on the path

find the successor of \(x_1\) in \(O(\log n)\) time

and the predecessor of \(x_2\) in \(O(\log n)\) time

which points in the tree should we include?

how do we do a lookup?

which points in the tree should we include?

Subtree decomposition

Warning: the root to split path isn't to scale

after the paths to \(x_1\) and \(x_2\) split...

any off-path subtree is either in or out

i.e. every point in the subtree has \(x_1 \leq x \leq x_2\) or none has

this will be useful for 2D range searching

1D range searching summary

lookup \((x_1, x_2)\) should report all points between \(x_1\) and \(x_2\)

preprocess \(n\) points on a line

\(O(n \log n)\) prep time

\(O(\log n + k)\) lookup time

where \(k\) is the number of points reported

(this is known as being output sensitive)

2D range searching

A 2D range searching data structure stores \(n\) distinct \((x, y)\)-pairs and supports:

the lookup \((x_1, x_2, y_1, y_2)\) operation

which returns every point in the rectangle \([x_1 : x_2] \times [y_1 : y_2]\)

i.e. every \((x, y)\) with \(x_1 \leq x \leq x_2\) and \(y_1 \leq y \leq y_2\).

Attempt one:

- Find all the points with \(x_1 \leq x \leq x_2\)
- Find all the points with \(y_1 \leq y \leq y_2\)
- Find all the points in both lists

How long does this take?

\[
O(\log n + k_x) + O(\log n + k_y) + O(k_x + k_y)
= O(\log n + k_x + k_y)
\]

these could be huge in comparison with \(k\)

here \(k_x\) is the number of points with \(x_1 \leq x \leq x_2\) (respectively for \(k_y\))
Subtree decomposition in 2D

How much space does our 2D range structure use?

How much prep time does our 2D range structure take?

Improving the query time

2D range searching
Improving the query time

The arrays of points at the children partition the array of the parent.

The arrays are sorted by y coordinate (but have been partitioned by x coordinate).

Consider a point in the parent array.

We add a link to its successor in both child arrays: (we do this for every point during preprocessing).

Observation: if we know where the successor of $y_1$ is in the parent, we know where it is in either child.

Adding these links doesn’t increase the space or the prep time.

The improved query time

How long does a query take?

The paths have length $O(\log n)$.

So steps 1. and 2. take $O(\log n)$ time.

As for step 3,

We do $O(\log n)$ 1D lookups...

Each takes $O(k')$ time.

This sums to...

$O(\log n + k)$

Query summary

1. Follow the paths to $x_1$ and $x_2$ (updating the successor to $y_1$ as you go).
2. Discard off-path subtrees where the $x$ coordinates are too large or too small.
3. For each off-path subtree where the $x$ coordinates are in range...

use the 1D range structure for that subtree to filter the $y$ coordinates.

Summary

$O(n \log n)$ prep time

$O(n \log n)$ space

$O(\log n + k)$ lookup time

where $k$ is the number of points reported.

we improved this :) using factional cascading.

2D range searching

A 2D range searching data structure stores $n$ distinct $(x, y)$ pairs and supports:

the lookup($x_1, x_2, y_1, y_2$) operation

which returns every point in the rectangle $[x_1 : x_2] \times [y_1 : y_2].$

i.e., every $(x, y)$ with $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2.