Computational Geometry

Chapter 5

Range Searching

Problem: Given a set of \( n \) points in \( \mathbb{R}^d \), preprocess them such that reporting or counting the \( k \) points inside a \( d \)-dimensional axis-parallel box will be most efficient.

Desired output-sensitive query time complexity – \( O(k + f(n)) \) for reporting and \( O(f(n)) \) for counting, where \( f(n) = o(n) \), e.g., \( f(n) = \log n \).

Sample application: Report all cities within 100 mile radius of Boston.

Range Searching – 1D

- In a one-dimensional world, points are real numbers and the query is two numbers (\( a, b \)).
- Simple \( O(\log n) \) algorithm:
  - Preprocessing: Sort points in \( O(n \log n) \).
  - Query: Binary search for \( a \) and \( b \) in list in \( O(\log n) \).
    - List all values inbetween.
- Cannot be easily generalized to higher dimensions (why not?).

Range Searching – 1D Tree

Range tree solution:

- Sort points.
- Construct a binary balanced tree, storing the points in its leaves.
- Each tree node stores the largest value of its left sub-tree.

Required time for finding a leaf: \( O(\log n) \).

Find the two boundaries of the given range in the leaves \( u \) and \( v \).

Report all the leaves in maximal subtrees between \( u \) and \( v \).

Mark the vertex at which the search paths diverge as V-split.

Continue to find the two boundaries, reporting values in the subtrees: When going left (right), report the entire right (left) subtree.

When reaching a leaf, check it exhaustively.

General Idea

- Build a data structure storing a "small" number of canonical subsets, such that:
  - The c.s. may overlap.
  - Every query may be answered as the union of a "small" number of c.s.
  - The geometry of the space enables this.

Two extremes:

- Singletons – \( O(1) \) query time, even for counting.
- Power set – \( O(2^n) \) query time. \( O(2^n) \) storage.
Given a set of points in 2D.
Bound the points by a rectangle.
Split the points into two (almost) equal size groups, using a horizontal or vertical line.
Continue recursively to partition the subsets, until they are small enough.
Canonical subsets are subtrees.

5. Partitions 2D space into axis-aligned rectangular regions.
Nodes represent partition lines, and leaves represent input points.
Label each node with the number of points in that rectangle.

6. Each node in the tree defines an axis-parallel rectangle in the plane, bounded by the lines marked by this vertex’s ancestors.
Label each node with the number of points in that rectangle.

7. Given an axis-parallel range query \( R \), search for this range in the tree.
Traverse only subtrees which represent regions intersecting \( R \).
If a subtree is contained entirely in \( R \):
\[ \text{Counting: Add its count.} \]
\[ \text{Reporting: Report entire subtree.} \]

Total query time: \( O(\sqrt{n} + k) \).

8. Given a set of points in 2D.
Bound the points by a rectangle.
Split the points into two (almost) equal size groups, using a horizontal or vertical line.
Continue recursively to partition the subsets, until they are small enough.
Canonical subsets are subtrees.

9. Given an axis-parallel range query \( R \), search for this range in the tree.
Traverse only subtrees which represent regions intersecting \( R \).
If a subtree is contained entirely in \( R \):
\[ \text{Counting: Add its count.} \]
\[ \text{Reporting: Report entire subtree.} \]

Total query time: \( O(\sqrt{n} + k) \).
For a d-dimensional space:
- Construction time: O(d n \log n).
- Space Complexity: O(d n).
- Query time complexity: O(d n^{1-1/d} + k).

Question: Are kd trees useful for non-orthogonal range queries, e.g., disks, convex polygons?

Fact: Using interval trees, orthogonal range queries may be solved in O(d n \log n) time and O(n) space.