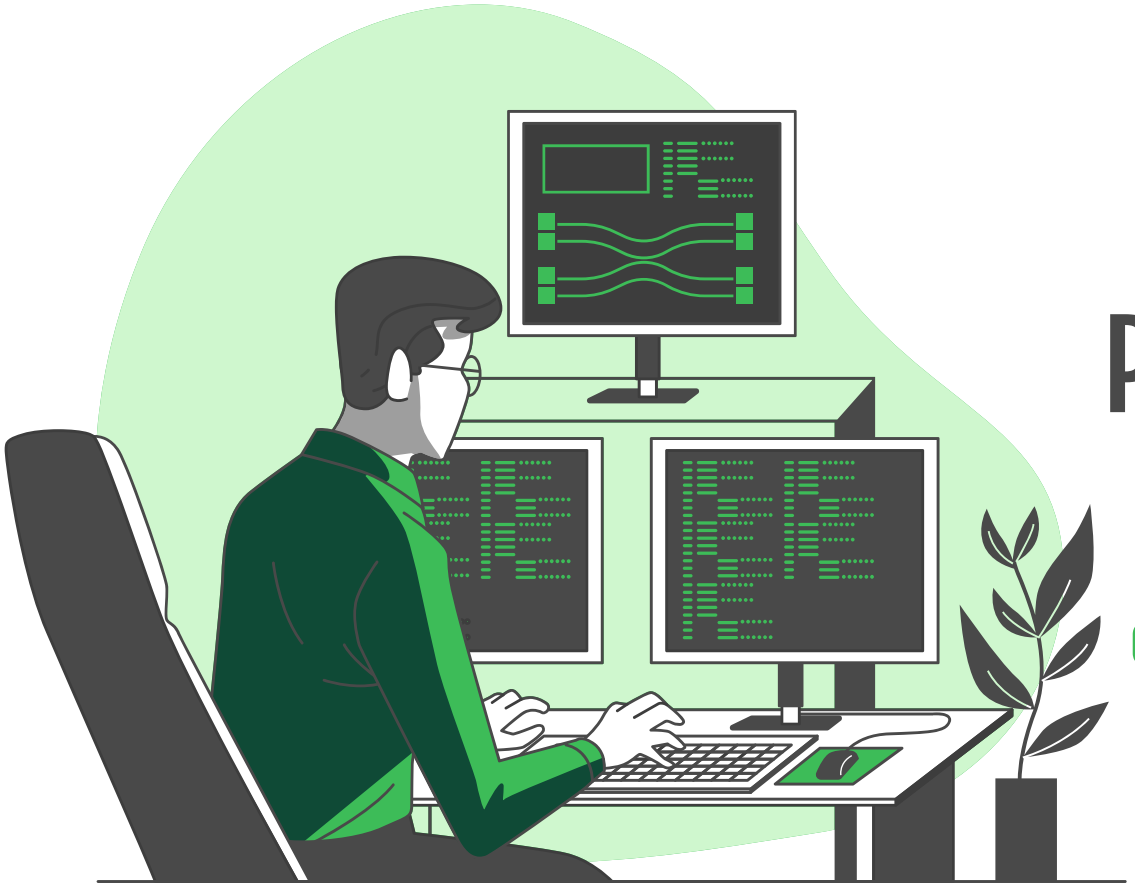
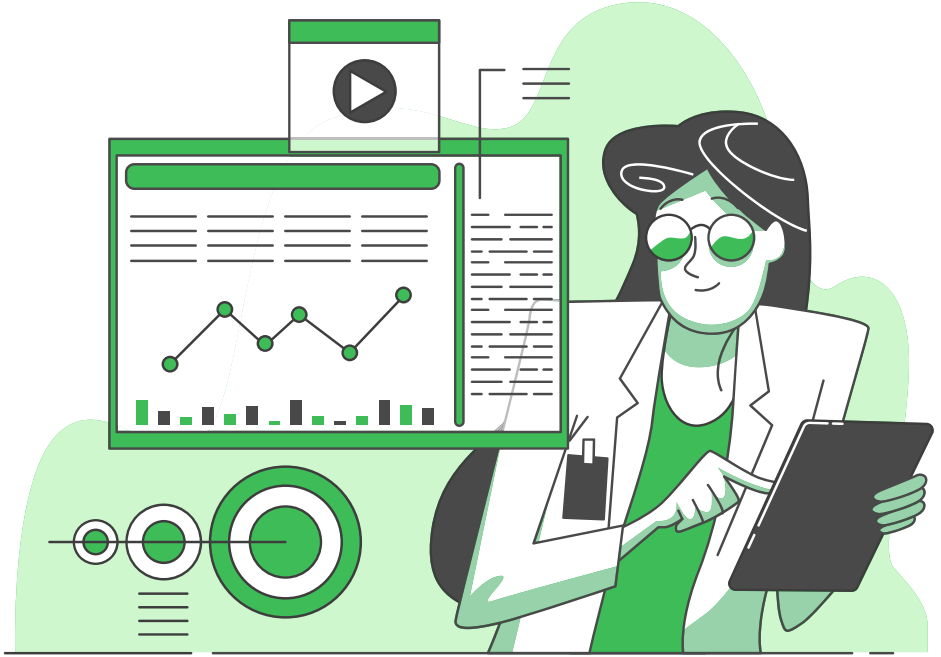


Point-Region Quadtree

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents



Data is Multidimensional

More Keys to Search

Every dimension in a data point can be a key.

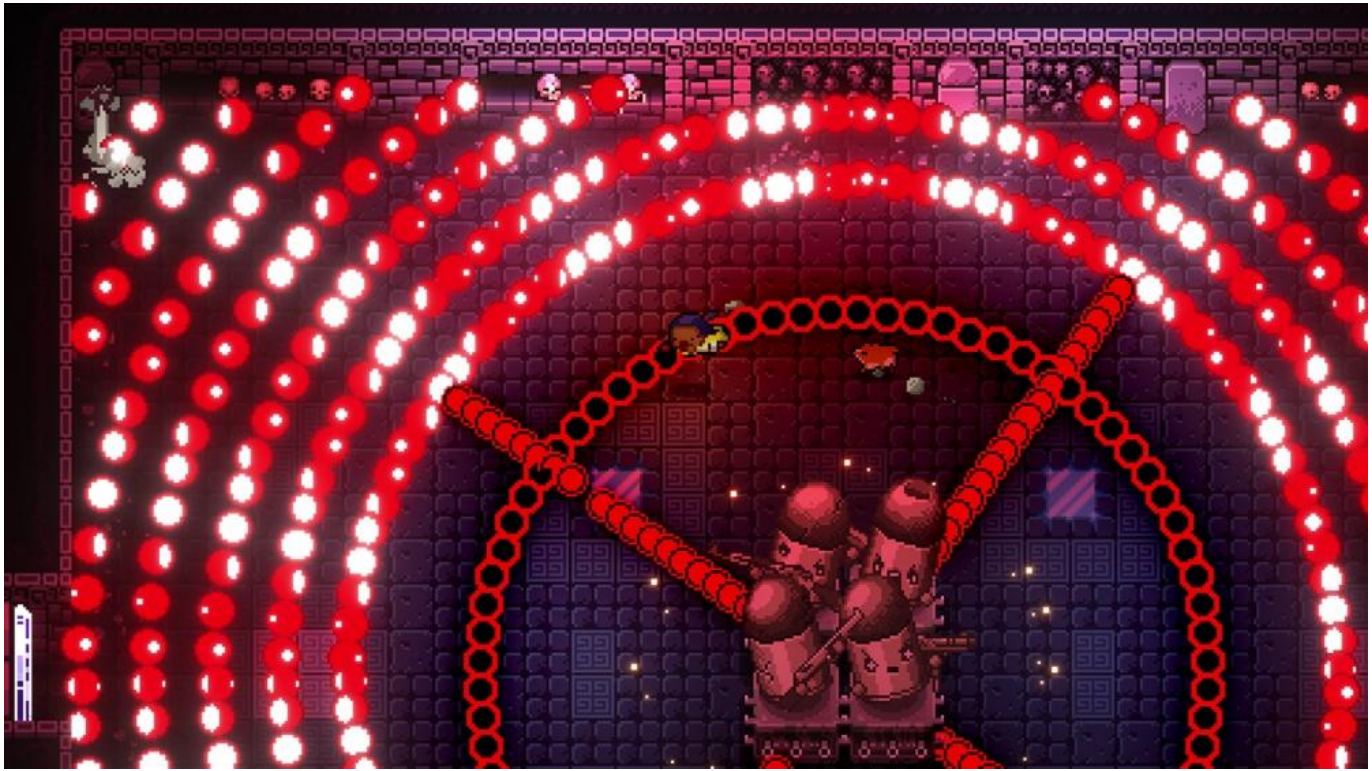
Geometric Data

Metric between data points is relevant.

Data Topology

Incidency between data points is relevant.

A Data Structure to Support This?



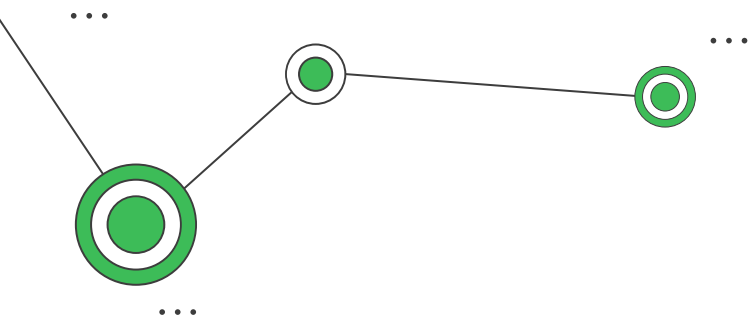
https://ec.nintendo.com/NZ/en/titles/70010000002861

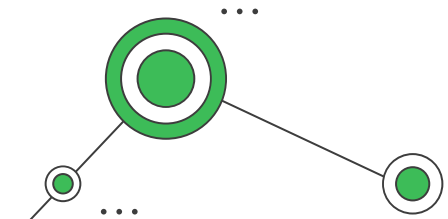
01

Point-Region Quadtree

Working with uniform planar subdivisions

Finkel, R. A., and J. L. Bentley. "Quad Trees: A Data Structure for Retrieval on Composite Keys." Acta Informatica 4, no. 1 (March 1974): 1-9.





https://doi.org/10.1007/BF00288933

Acta Informatica 4, 1—9 (1974)

Quad Trees
A Data Structure for Retrieval on Composite Keys

R. A. Finkel and J. L. Bentley

Received April 8, 1974

Summary. The quad tree is a data structure appropriate for storing information to be retrieved on composite keys. We discuss the specific case of two-dimensional retrieval, although the structure is easily generalised to arbitrary dimensions. Algorithms are given both for straightforward insertion and for a type of balanced insertion into quad trees. Empirical analyses show that the average time for insertion is logarithmic with the tree size. An algorithm for retrieval within regions is presented along with data from empirical studies which imply that searching is reasonably efficient. We define an optimized tree and present an algorithm to accomplish optimization in $n \log n$ time. Searching is guaranteed to be fast in optimized trees. Remaining problems include those of deletion from quad trees and merging of quad trees, which seem to be inherently difficult operations.

Introduction

One way to attack the problem of retrieval on composite keys is to consider records arranged in a several-dimensional space, with one dimension for every attribute. Then a query concerning the presence or absence of records satisfying given criteria becomes a specification of some (possibly disconnected) subset of that space. All records which lie in that subset are to be returned as the response to the query.

The retrieval of information on only one key has been well studied. Experience has shown binary trees serve as a good data structure for representing linearly ordered data, and that balanced binary trees provide a guaranteed fast structure (Knuth, 6.2.3).

This paper will discuss a generalization of the binary tree for the treatment of data with inherently two-dimensional structure. One clear example of such records is that of cities on a map. A sample query might be: "Find all the cities which are within 300 miles of Chicago or north of Seattle." The data structure we propose to handle such queries is called a quad tree. It will be obvious that the basic concepts involved are easily generalized to records of any dimensionality.

Definitions and Notation

The location of records with two-dimensional keys will be stored in a tree with out-degree four at each node. Each node will store one record and will have up to four sons, each a node. The root of the tree divides the universe into four quadrants, namely NE, NW, SW, and SE (using the map analogy). Let us call these quadrants one, two, three and four, respectively. Fig. 1 shows the correspondence between a simple tree and the records it represents.

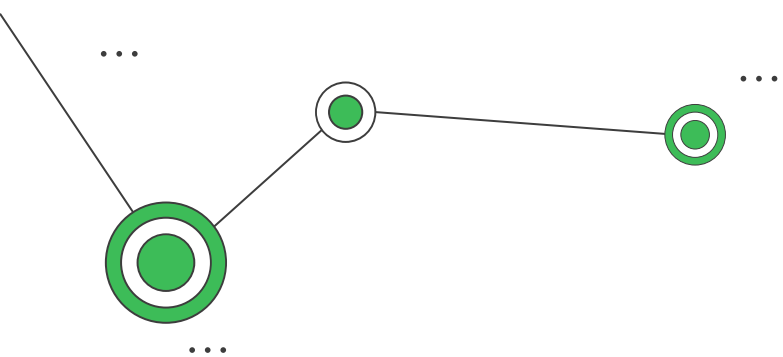
The convention we use for points which lie directly on one of the quadrant lines emanating from a node is as follows: Quadrants one and three are closed,

Quadtrees

Quadtrees are tree data structures where each node has exactly four children.

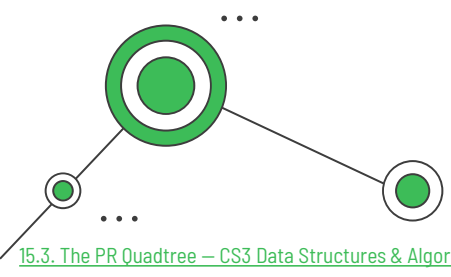
Useful to partition a two-dimensional space by recursive subdivisions into four quadrants.

Each leaf represents a “unit of interesting spatial information”.



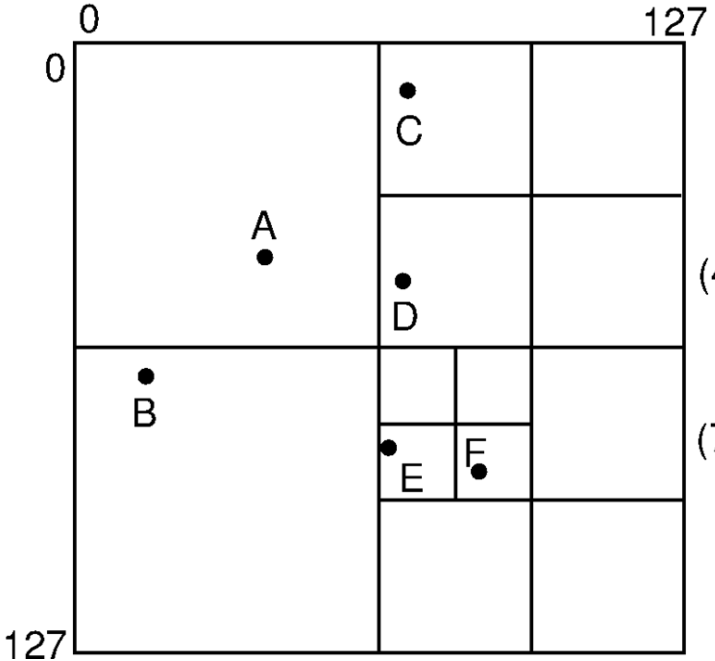
Point-Region Quadtree

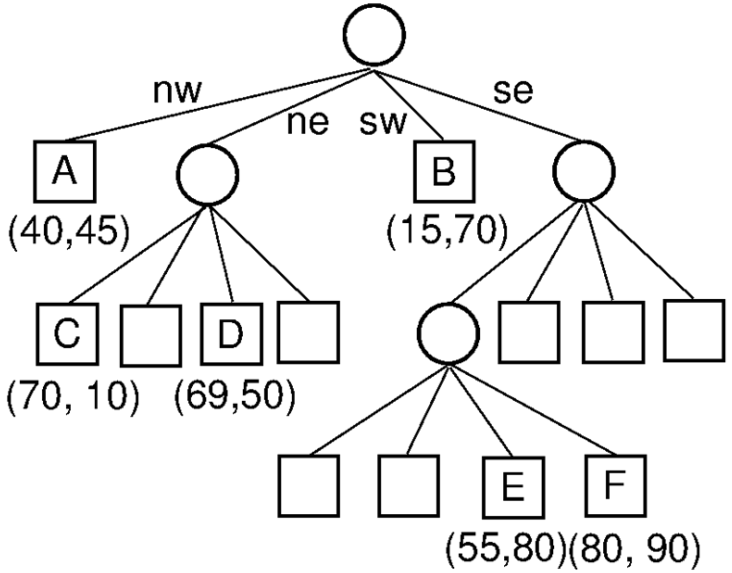
- Each node either has exactly four children (NW, NE, SW, SE) or is a leaf (stores at most one data point).
- Full four-way branching (4-ary) tree in shape.
- Uniform subdivision of each (sub)quadrant continues until no leaf contains more than a single point.
- Useful to search individual points or regions in the plane.



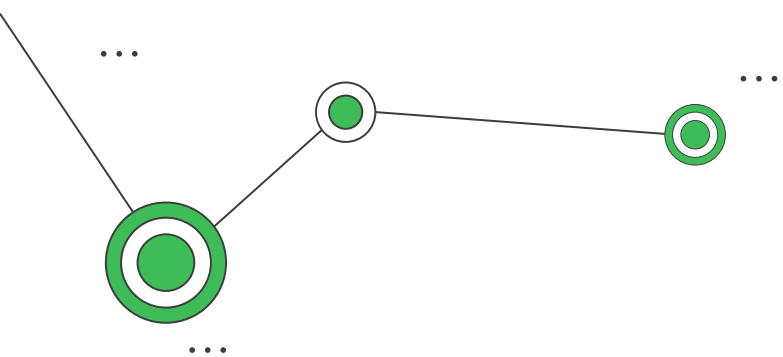
15.3. The PR Quadtree — CS3 Data Structures & Algorithms (vt.edu)

Point-Region Quadtree Example





15.3. The PR Quadtree – CS3 Data Structures & Algorithms (vt.edu)



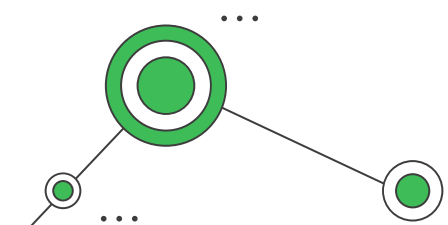
Point-Region Quadtree Node

Each node stores:

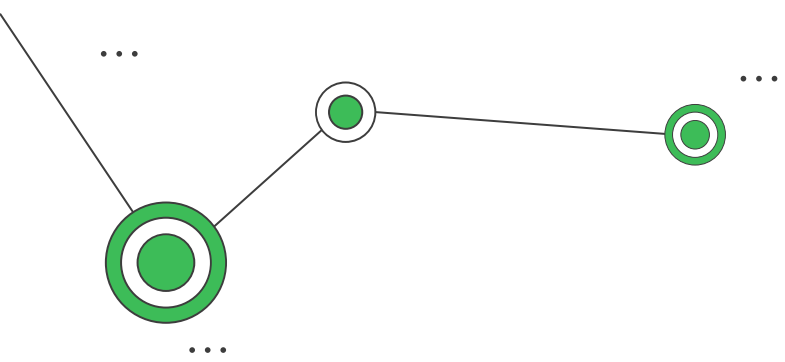
- The rectangular region it represents.
- A 2D point, or
- Quadrants.

Let **n** be a Point-Region Quadtree node, then:

- **n.region** refers to the tuple storing the minimum and maximum coordinates of the region.
- **n.point** refers to the 2D point the node stores.
- **n.children** refers to the container with the memory references to four Point-Region Quadtree nodes: NW, NE, SW, and SE.

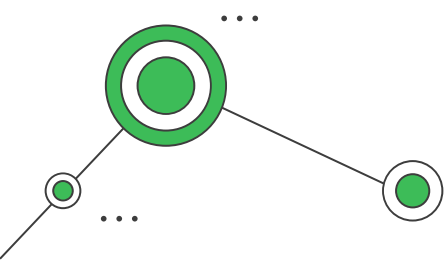


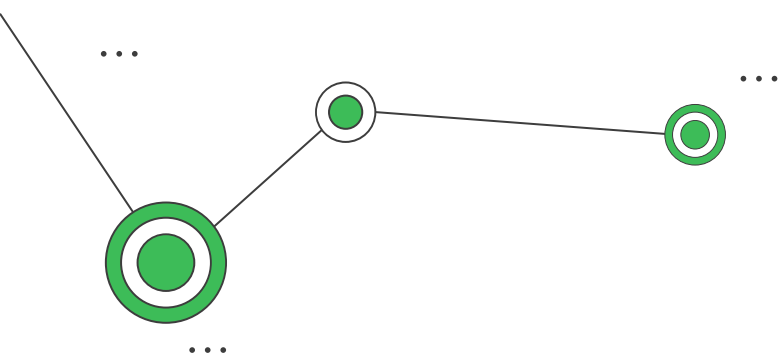
Note: This description supports at most one 2D point per leaf. Yet, more than one point can be store, provided we control the maximum capacity. Once the points overflow, the node must be split into quadrants.



In Boundary and Subdivide

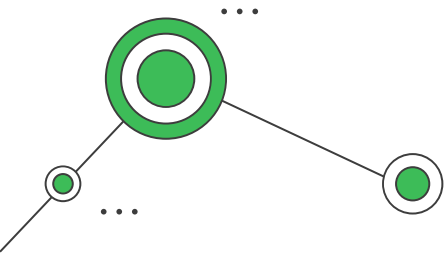
```
algorithm inboundary(B:region, P:point) → bool
   return B.xmin ≤ P.x < B.xmax and
   B.ymin ≤ P.y < B.ymax
end algorithm


algorithm subdivide(root:node)
   xmin, ymin, xmax, ymax ← root.region
   xmid ← (xmin + xmax) / 2
   ymid ← (ymin + ymax) / 2
   root.children ← [
      Node(Region(xmin, ymid, xmid, ymax)), //NW
      Node(Region(xmid, ymid, xmax, ymax)), //NE
      Node(Region(xmin, ymin, xmid, ymid)), //SW
      Node(Region(xmid, ymin, xmax, ymid))] //SE
   if root.point is not Null then
      P ← root.point
      root.point ← Null
      for each quadrant in root.children do
         if insert(P, quadrant) then
            return
         end if
      end for
   end if
end algorithm
```





Insert

```
algorithm insert(P:point, root:node) → bool
   if not inboundary(root.region, P) then
      return false
   end if
   if root.point is Null and |root.children| = 0 then
      root.point ← P
      return true
   end if
   if |root.children| = 0 then
      subdivide(root)
   end if
   for each quadrant in root.children do
      if insert(P, quadrant) then
         return true
      end if
   end for
end algorithm
```



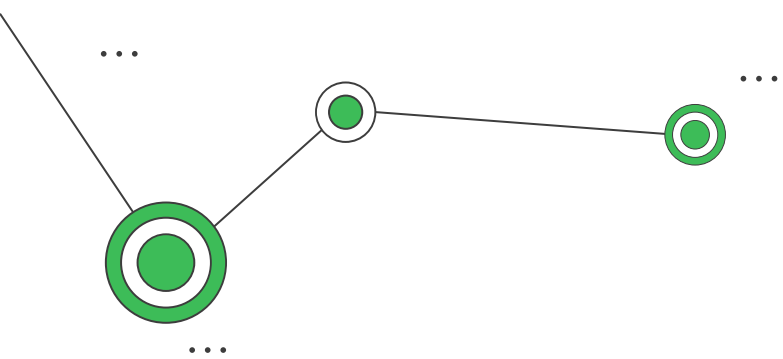
Search In Point-Region Quadtrees

Single point search: Let P be a point in a Point-Region Quadtree.

- If the root is a leaf, check if the root's data matches with P.
- Otherwise, continue the search recursively to the quadrant that contains P.

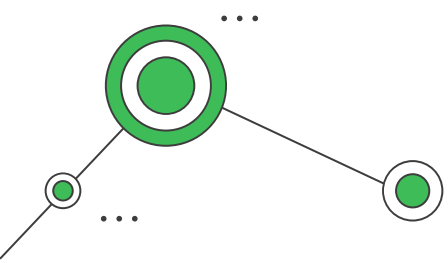
Region search: To locate all points within radius r of query point P:

- Begin at the root. If the root is an empty leaf node, then no data points are found.
- If the root is a leaf containing a data record, then the location of the data point is examined to determine if it falls within the circle.
- If the root is an internal node, then the process is performed recursively, but only on those subtrees containing some part of the search circle.



Range Query

```
algorithm overlaps(R1:region, R2:region) → bool
   xmin1, ymin1, xmax1, ymax1 ← R1
   xmin2, ymin2, xmax2, ymax2 ← R2
   return not (xmax1 ≤ xmin2 or xmin1 ≥ xmax2 or
               ymax1 ≤ ymin2 or ymin1 ≥ ymax2)
end algorithm


algorithm rangequery(range:region, root:node)
   let results be an empty container
   if not overlaps(range, root.region) then
      return results
   end if
   if root.point is not Null and inboundary(range, root.point) then
      results.append(root.point)
   end if
   if |root.children| = 0 then
      return results
   end if
   for each quadrant in root.children do
      result ← rangequery(range, quadrant)
      if |result| > 0 then
         results.join(result)
      end if
   end for
   return results
end algorithm
```



Search In Point-Region Quadtrees (cont.)

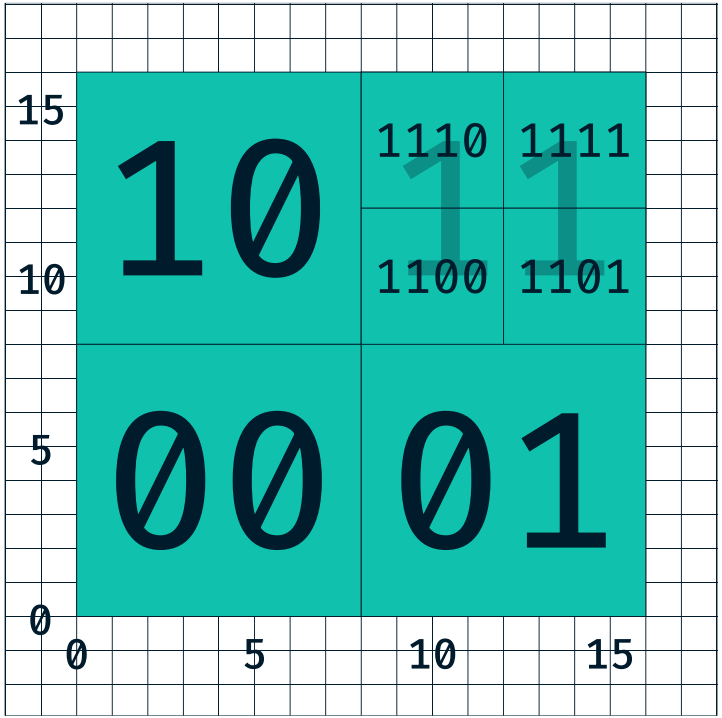
Search complexity?

- Balanced? $O(\log_4(n))$
- Not balanced? $O(n)$

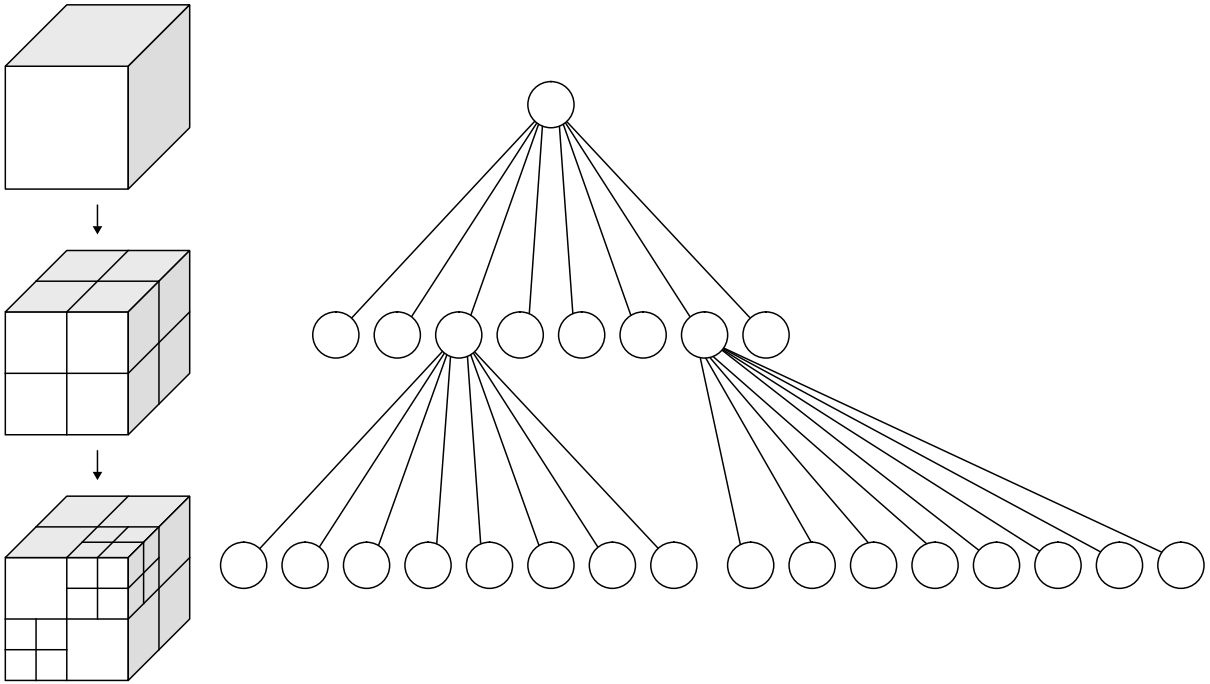
Balancing challenges:

- Uniform subdivisions? Most likely not balanced.
- Requires adjustable subdivisions.
- Dense regions require more subdivisions.

Recursive Binary Space Partitioning QT



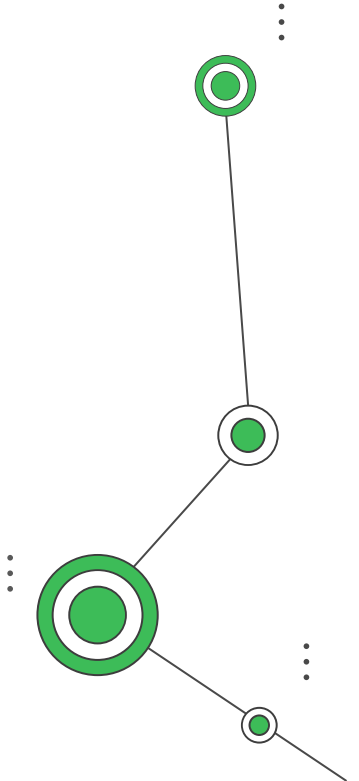
Octree (For 3D Points)

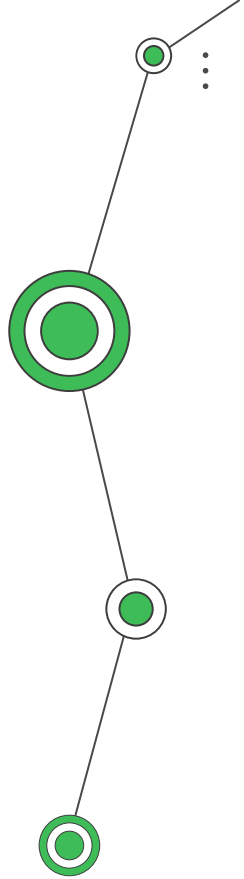


By WhiteTimberwolf, PNG version: Nü - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=9851485

Done!

Do you have any questions?





CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories