spatial tree algorithms
learning objectives

- learn the characteristics of spatial data
- learn several spatial indexing data structures
- learn basic algorithms for using such structures
computational geometry

A branch of computer science focusing on data structures & algorithms for solving geometric problems

development made possible by exponential progress in computer graphics, with multiple applications

mathematical visualization, e.g., proof without words, mandelbrot sets

gamegraphic information systems, e.g., location search & route planning

computer vision e.g., 3D graphics is games

computer-aided engineering, e.g., mechanical design
computational geometry

what's specific to spatial data?

with 1-dimensional data, natural ordering implicitly partitions the data, e.g., binary tree

spatial data is intrinsically multidimensional, so there is no natural ordering of data (e.g., of points)

with 1-dimensional data, the static case is rather simple and solved by sorting the data

with multidimensional data, the static case is far from simple and solved by several partitioning techniques
computational geometry

typical problems

**nearest neighbor**: given a set of points $P$, find which one is closest to a target point $p_t$

**range queries**: given a set of points $P$, find the points contained within a given rectangle

**intersection queries**: given a set of rectangles $R$, find which rectangles intersect a target rectangle

**collision detection**: given a set of shapes $S$, find the intersections between all these shapes
computational geometry

**typical approaches**

**brute-force algorithm**

**nearest neighbor:** given a set of points $P$, find which one is closest to a target point $p_t$

Complexity: $O(n)$, with $n = |P|$

**Nearest–neighbor** $(P, p_t)$

```plaintext
p ← NIL
min ← ∞
for each $p_i ∈ P$
    if distance$(p_i, p_t) < min$
        min ← distance$(p_i, p_t)$
        $p ← p_i$
return $(p, min)$
```

**spatial tree structures**

they index spatial objects

**Complexity:** $O(\log n)$, with $n = |P|$

- **quad-trees**
- **kd-trees**
- **R-trees**
A recursive tree, where each node has between $M$ and $m = \left\lfloor \frac{M}{2} \right\rfloor$ children, except for the root which has at least two children.

Only leaf nodes contain actual spatial object entries, each consisting of the spatial object itself and a minimum bounding region (mbr) containing that object, i.e., $\text{object} = (\text{shape}, \text{mbr})$.

Internal nodes contain children entries, each consisting of a link to the child node and an mbr covering all children nodes of that child, i.e., $\text{node} = (\text{child}, \text{mbr})$.

An minimum bounding region is typically of the form $\text{mbr} = (x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$.

All leaves are at the same level, i.e., the tree is height balanced.
R-tree

only leaf nodes contain actual spatial object entries, each consisting of the spatial object itself and a minimum bounding region (mbr) containing that object, i.e., \( \text{object} = (\text{shape}, \text{mbr}) \)

internal nodes contain children entries, each consisting of a link to the child node and an mbr covering all children nodes of that child, i.e., \( \text{node} = (\text{child}, \text{mbr}) \)

important: the root also contains a minimum bounding box
**R-tree**

**INTERSECT** \((\text{node}, \text{region})\)

- **if** \(\text{node.mbr} \subset \text{region}\)
  - return \(\{ \text{object} \mid \text{object} \in \text{REACHABLE-LEAVES}(\text{node}) \}\)

- **if** \(\text{node is a leaf}\)
  - return \(\{ \text{object} \in \text{node} \mid \text{object.mbr} \cap \text{region} \neq \emptyset \}\)

  \(\text{result} \leftarrow \emptyset\)

  **for each** \(\text{kid} \in \text{node.children}\)

  - **if** \(\text{kid.mbr} \cap \text{region} \neq \emptyset\)
    - \(\text{result} = \text{result} \cup \text{INTERSECT}(\text{kid.child}, \text{region})\)

  return \(\text{result}\)

**SEARCH** \((\text{node}, \text{shape})\)

- **if** \(\text{node is a leaf}\)

  - **if** \(\exists \text{object} \in \text{node} : \text{object.shape} = \text{shape}\)
    - return \(\text{object}\)

  - return \(\text{NIL}\)

  **for each** \(\text{kid} \in \text{node.children}\)

  - **if** \(\text{shape.mbr} \subseteq \text{kid.mbr}\)
    - return \(\text{SEARCH(kid.child, shape)}\)

- return \(\text{NIL}\)

**important:**
the root also contains a minimum bounding box
quad-tree

a recursive tree where each internal node has four children

each node represents a cell in the geometrical space, with its children partitioning that cell into an equally sized subcell

predefined partitioning with subcells (quadrants) named as North West (NW), North-East (NE), South-West (SW) and South-East (SE)

like R-trees, only leaf nodes store actual geometrical objects

quad-tree

region quad-tree

point-region quad-tree
quad-tree

region quad-tree

point-region quad-tree
ADD (node, point)
    if point \notin node.cell
        return FALSE
    if node is a leaf
        if node.point \in region  return \{ node.point \}
        return \Ø
    if node.point = NIL
        node.point \leftarrow point
        return TRUE

quadrant \leftarrow FIND-QUADRANT(node, point)
if node is a leaf
    SUBDIVIDE(node)
return ADD (node[quadrant], point)

INTERSECT (node, region)
    if node is a leaf
        if node.point \in region  return \{ node.point \}
        return \Ø
    if node.cell \subset region
        return \{ node.point \mid node \in REACHABLE-LEAVES(node) \}
    result \leftarrow \Ø
    for each quadrant \in \{ NW, NE, SW, SE \}
        if node[quadrant].cell \cap region \neq \Ø
            result = result \cup INTERSECT (node[quadrant], region)
    return result

root
NW
SE
NE
SW
A *kd-tree* (short for *k*-dimensional tree) is a binary tree in which every node is a *k*-dimensional point. In addition, each internal node divides the *k*-dimensional space into two parts known as half-spaces. All points in one half space are contained in the left subtree of the node and all points in the other half space contained in the right subtree. All nodes at the same level (height) divide the *k*-dimensional space according to the same cutting dimension (axis).

ADD (node, point, cutaxis)
    if node = NIL
        node ← CREATE-NODE
        node.point = point
        return node
    if point[cutaxis] ≤ node.point[cutaxis]
        node.left = ADD(node.left, point, (cutaxis + 1) mod k)
    else
        node.right = ADD(node.right, point, (cutaxis + 1) mod k)
    return node

Remarks
• Points are stored as k-dimensional arrays
• Each axis corresponds to an index:
  ▸ x-axis corresponds to index 0
  ▸ y-axis corresponds to index 1
  ▸ etc...
• So assuming point p_i = (x_i,y_i) = (3,7),
  we have that p_i = [3,7], x_i = p[0] = 7
  and y_i = p[1] = 7
• In this example, initially root = NIL
  and points are inserted as follows:
  ▸ ADD(root,p_1, 0)
  ▸ ADD(root,p_2, 0)
  ▸ ADD(root,p_3, 0)
  ▸ etc...