Kd-Trees

Outline:

I. Construction of a kd-tree

II. Nodes and regions

III. Query using a Kd-tree

IV. Analysis of query time
I. 2D Range Search

Point set: $P$
I. 2D Range Search

Point set: \( P \)

Assumptions (removable);
- No two points have the same \( x \)-coordinate.
- No two points have the same \( y \)-coordinate.
I. 2D Range Search

Point set: $P$

Assumptions (removable);
- No two points have the same $x$-coordinate.
- No two points have the same $y$-coordinate.

Query rectangle: $[x, x'] \times [y, y']$
I. 2D Range Search

Point set: $P$

Assumptions (removable):

- No two points have the same $x$-coordinate.
- No two points have the same $y$-coordinate.

Query rectangle: $[x, x'] \times [y, y']$
I. 2D Range Search

Point set: $P$

Assumptions (removable);
- No two points have the same $x$-coordinate.
- No two points have the same $y$-coordinate.

Query rectangle: $[x, x'] \times [y, y']$

# answer points may be $\ll |P|$. 
I. 2D Range Search

Point set: $P$

Assumptions (removable):
- No two points have the same $x$-coordinate.
- No two points have the same $y$-coordinate.

Query rectangle: $[x, x'] \times [y, y']$

# answer points may be $\ll |P|$.

Many queries.
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree

\( p_1 \)

\( l_2 \)

\( p_2 \)

\( l_5 \)

\( p_3 \)

\( l_1 \)

\( p_5 \)

\( p_7 \)

\( p_8 \)

\( p_9 \)

\( p_4 \)

\( l_4 \)

\( p_6 \)

\( p_{10} \)
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree
Generation of a Kd-Tree
Strategy

- Split the point set $P$ equally with a vertical line $l_1$.
  
  Store the splitting line $l_1$ at the root.
Strategy

- Split the point set $P$ equally with a **vertical** line $l_1$.
  
  Store the splitting line $l_1$ at the root.

- Split the left subset of points equally with a **horizontal** line $l_2$. 
Strategy

- Split the point set $P$ equally with a **vertical** line $l_1$.
  
  Store the splitting line $l_1$ at the root.

- Split the left subset of points equally with a **horizontal** line $l_2$.
  
  Split the right subset of points equally with a **horizontal** line $l_3$. 
Strategy

- Split the point set $P$ equally with a vertical line $l_1$.
  
  Store the splitting line $l_1$ at the root.

- Split the left subset of points equally with a horizontal line $l_2$.
  Split the right subset of points equally with a horizontal line $l_3$.

- Split the four subsets vertically again.

...
Kd-Tree

l_1

l_2

l_3

l_4

l_5

l_6

l_7

l_8

p_1

p_2

p_3

p_4

p_5

p_6

p_7

p_8

p_9

p_10
Kd-Tree

Diagram of a Kd-Tree with levels and points.
Splitting

At the *median*: $\left\lfloor n/2 \right\rfloor$th smallest number.
Splitting

At the median: \([n/2]\) th smallest number.

- Vertical split by \(l\)
  \[
  \{ p \mid p \text{ on or to the left of } l \} \cup \{ p \mid p \text{ to the right of } l \}
  \]
Splitting

At the median: \( \left\lfloor n/2 \right\rfloor \) th smallest number.

- Vertical split by \( l \)
  
  \[
  \{ p \mid p \text{ on or to the left of } l \}\]
  \[
  \cup \{ p \mid p \text{ to the right of } l \}\]

- Horizontal split by \( l \)
  
  \[
  \{ p \mid p \text{ on or below } l \}\]
  \[
  \cup \{ p \mid p \text{ above } l \}\]
Tree Construction

BuildKdTree\((P,d)\)

\[\begin{align*}
\text{if } |P| = 1 & \quad \text{then return a leaf} \\
\text{else if } d \text{ even} & \quad \text{then split } P \text{ with a vertical line } l \text{ through the median } x\text{-coordinate into } P_1 \text{ and } P_2 \\
\text{else } & \quad \text{split } P \text{ with a horizontal line } l \text{ through the median } y\text{-coordinate into } P_1 \text{ and } P_2 \\
\end{align*}\]

\[\begin{align*}
v_{\text{left}} & \leftarrow \text{BuildKdTree}(P_1, d + 1) \\
v_{\text{right}} & \leftarrow \text{BuildKdTree}(P_2, d + 1)
\end{align*}\]
Tree Construction

BuildKdTree(P,d)

if |P| = 1
    then return a leaf
else if d even
    then split P with a vertical line l through the median x-coordinate into P₁ and P₂
else split P with a horizontal line l through the median y-coordinate into P₁ and P₂

v_left ← BuildKdTree(P₁, d + 1)
v_right ← BuildKdTree(P₂, d + 1)
Presorting

For efficiency of splitting, maintain $n$ points in two lists:

- sorted on $x$-coordinate
- sorted also on $y$-coordinate

\[(2, 10) \rightarrow (4, 3) \rightarrow (7, 5)\]
Building Time

\[ T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
? & \text{if } n > 1 \\
O(n) + 2T\left(\frac{n}{2}\right) & \text{if } n > 1 
\end{cases} \]
Building Time

\[ T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
? & \\
O(n) + 2T\left(\frac{n}{2}\right) & \text{if } n > 1
\end{cases} \]

\[ T(n) = O(n \log n) \]
Building Time

\[ T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
? & \\
O(n) + 2T\left(\frac{n}{2}\right) & \text{if } n > 1 
\end{cases} \]

\[ T(n) = O(n \log n) \]

Subsumes sorting time!
II. Root of the Kd-Tree
Internal Node

Plane $l_1$

Left half-plane $l_2$
(cont’d)
Node as a Rectangular Region

$l_1$, $l_2$, $l_3$

plane, left half-plane, lower left quarter

$T(v)$

region$(v)$
Each node $v$ represents a rectangular region bounded by splitting lines stored at ancestors of $v$.

This region is split by the line storing at $v$. 
How to Locate the Region?

Path: root $\rightarrow v$
- Left of $l_1$
- Below $l_2$
- Right of $l_3$
III. Range Searching

**Strategy**: Search the subtree rooted at $v$ only if the query rectangle $R$ intersects region($v$).
III. Range Searching

**Strategy:** Search the subtree rooted at $\nu$ only if the query rectangle $R$ intersects region($\nu$).

- region($\nu$) $\subseteq$ $R$

  Report all leaves (points) of $T(\nu)$. 

![Diagram showing region($\nu$) and R intersecting]
III. Range Searching

**Strategy:** Search the subtree rooted at $v$ only if the query rectangle $R$ intersects $\text{region}(v)$.

- $\text{region}(v) \subseteq R$
  - Report all leaves (points) of $T(v)$.

- $\text{region}(v) \not\subseteq R$ and $\text{region}(v) \cap R \neq \emptyset$
  - Recursively check $\text{region}(\text{lchild}(v))$ and $\text{region}(\text{rchild}(v))$. 
(cont’d)
(cont’d)
Search Algorithm

SearchKdTree(\(v, R\))

1. if \(v\) is a leaf
2. then if \(v \in R\)
3. then report \(v\)
4. else if region(lchild(\(v\))) \(\subseteq R\)
5. then ReportSubtree(lchild(\(v\))) \(// O(\# \text{ leaves})\)
6. else if region(lchild(\(v\))) \(\cap R \neq \emptyset\)
7. then SearchKdTree(lchild(\(v\)), \(R\))
8. if region(rchild(\(v\))) \(\subseteq R\)
9. then ReportSubtree(rchild(\(v\))) \(// O(\# \text{ leaves})\)
10. else if region(rchild(\(v\))) \(\cap R \neq \emptyset\)
11. then SearchKdTree(rchild(\(v\)), \(R\))
Search Algorithm

SearchKdTree($v, R$)

1. if $v$ is a leaf
2. then if $v \in R$
3. then report $v$
4. else if region(lchild($v$)) $\subseteq R$
5. then ReportSubtree(lchild($v$)) // $O$(# leaves)
6. else if region(lchild($v$)) $\cap R \neq \emptyset$
7. then SearchKdTree(lchild($v$), $R$)
8. if region(rchild($v$)) $\subseteq R$
9. then ReportSubtree(rchild($v$)) // $O$(# leaves)
10. else if region(rchild($v$)) $\cap R \neq \emptyset$
11. then SearchKdTree(rchild($v$), $R$)

Current region is maintained through recursive calls:
Search Algorithm

SearchKdTree(\(v, R\))

1. if \(v\) is a leaf
2. then if \(v \in R\)
3. then report \(v\)
4. else if region(lchild(\(v\))) \(\subseteq R\)
5. then ReportSubtree(lchild(\(v\))) // \(O(\# \text{ leaves})\)
6. else if region(lchild(\(v\))) \(\cap R \neq \emptyset\)
7. then SearchKdTree(lchild(\(v\)), \(R\))
8. if region(rchild(\(v\))) \(\subseteq R\)
9. then ReportSubtree(rchild(\(v\))) // \(O(\# \text{ leaves})\)
10. else if region(rchild(\(v\))) \(\cap R \neq \emptyset\)
11. then SearchKdTree(rchild(\(v\)), \(R\))

Current region is maintained through recursive calls:
- \(l\): line stored at node \(v\)
Search Algorithm

SearchKdTree(v, R)

1. if v is a leaf
2. then if v ∈ R
3. then report v
4. else if region(lchild(v)) ⊆ R
5. then ReportSubtree(lchild(v)) // O(# leaves)
6. else if region(lchild(v)) ∩ R ≠ ∅
7. then SearchKdTree(lchild(v), R)
8. if region(rchild(v)) ⊆ R
9. then ReportSubtree(rchild(v)) // O(# leaves)
10. else if region(rchild(v)) ∩ R ≠ ∅
11. then SearchKdTree(rchild(v), R)

Current region is maintained through recursive calls:

- l: line stored at node v
- region(lchild(v)) = region(v) ∩ half-plane left of or below l
IV. Time Analysis – Count Node Visits

Determine the *number* of visited nodes.

**Type 1**

- **Nodes reported as a traversal of some subtree** (lines 3, 5 & 9)

1. if $v$ is a leaf
2. then if $v \in R$
3. then report $v$
4. else if region(lchild($v$)) $\subseteq R$
5. then ReportSubtree(lchild($v$))
6. else if region(lchild($v$)) $\cap R \neq \emptyset$
7. then SearchKdTree(lchild($v$), $R$)
8. if region(rchild($v$)) $\subseteq R$
9. then ReportSubtree(rchild($v$))
10. else if region(rchild($v$)) $\cap R \neq \emptyset$
11. then SearchKdTree(rchild($v$), $R$)
IV. Time Analysis – Count Node Visits

Determine the *number* of visited nodes.

**Type 1**

- Nodes reported as a *traversal of some subtree* (lines 3, 5 & 9)

1. if \( v \) is a leaf
2. then if \( v \in R \)
3. then report \( v \)
4. else if region(lchild(\( v \))) \( \subseteq R \)
5. then ReportSubtree(lchild(\( v \)))
6. else if region(lchild(\( v \))) \( \cap R \neq \emptyset \)
7. then SearchKdTree(lchild(\( v \)), \( R \))
8. if region(rchild(\( v \))) \( \subseteq R \)
9. then ReportSubtree(rchild(\( v \)))
10. else if region(rchild(\( v \))) \( \cap R \neq \emptyset \)
11. then SearchKdTree(rchild(\( v \)), \( R \))

*Linear* in the number of reported points.
Type 2 Nodes

Type 2

Nodes visited but not in a traversed subtree
(lines 1, 6-7, 10-11)

1. if $v$ is a leaf
2. then if $v \in R$
3. then report $v$
4. else if region(lchild($v$)) $\subseteq R$
5. then ReportSubtree(lchild($v$))
6. else if region(lchild($v$)) $\cap R \neq \emptyset$
7. then SearchKdTree(lchild($v$), $R$)
8. if region(rchild($v$)) $\subseteq R$
9. then ReportSubtree(rchild($v$))
10. else if region(rchild($v$)) $\cap R \neq \emptyset$
11. then SearchKdTree(rchild($v$), $R$)
Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree
  (lines 1, 6-7, 10-11)
Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node $\nu$ satisfies
Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node $v$ satisfies

\[
\text{region}(v) \cap R \neq \emptyset \\
\text{region}(v) \not\subseteq R
\]
Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node $v$ satisfies

  \[ \text{region}(v) \cap R \neq \emptyset \]
  \[ \text{region}(v) \not\subseteq R \]

\[ \downarrow \]

\[ \# \text{ type 2 nodes} \leq \# \text{ regions intersected by the four edges of } R \]
Type 2 Nodes

Type 2

- Nodes visited but not in a traversed subtree (lines 1, 6-7, 10-11)

- Each such node $v$ satisfies

  \[ \text{region}(v) \cap R \neq \emptyset \]
  \[ \text{region}(v) \not\subset R \]

\[ \downarrow \]

# type 2 nodes $\leq$ # regions intersected by the four edges of $R$

- How to bound #regions intersected by a vertical line?
Setting up a Recurrence

\[ Q(n) \]: number of intersected regions in a kd-tree which
- stores \( n \) points, and
- has a vertical line \( l \) as the root.

Consider a vertical edge (as a line) of the query box.
Setting up a Recurrence

\( Q(n) \): number of intersected regions in a kd-tree which
- stores \( n \) points, and
- has a vertical line \( l \) as the root.

Consider a vertical edge (as a line) of the query box.

A vertical line \( m \) intersects one side of \( l \), say, its left side.
Setting up a Recurrence

\(Q(n)\): number of intersected regions in a kd-tree which
- stores \(n\) points, and
- has a vertical line \(l\) as the root.

Consider a vertical edge (as a line) of the query box.

A vertical line \(m\) intersects one side of \(l\), say, its left side.

The region corresponds to a horizontal splitting line \(l'\).
Setting up a Recurrence

\[ Q(n) \]: number of intersected regions in a kd-tree which
- stores \( n \) points, and
- has a vertical line \( l \) as the root.

Consider a vertical edge (as a line) of the query box.

A vertical line \( m \) intersects one side of \( l \), say, its left side.

The region corresponds to a horizontal splitting line \( l' \).

\( m \) intersects both children of \( l' \), whereas it intersects only one of \( l \).
Setting up a Recurrence

\( Q(n) \): number of intersected regions in a kd-tree which
- stores \( n \) points, and
- has a vertical line \( l \) as the root.

Consider a vertical edge (as a line) of the query box.

A vertical line \( m \) intersects one side of \( l \), say, its left side.

The region corresponds to a horizontal splitting line \( l' \).

\( m \) intersects both children of \( l' \), whereas it intersects only one of \( l \).

Not a recurrence situation!
Recurrence (cont’d)

Go down two levels!

- Four nodes at depth 2
- Each corresponds to $n/4$ points.
- Only two represent intersected regions.

```
 m  l'''  l
 l'    n/4 points
 l''  n/4 points
 l    n/2 points
```

```
 Q(n)
 /
 Q(n/4) l' Q(n/4)
 /
 l'' l'''
 /
```

```
 v
 /
```
Running Time

\[ Q(n) = \begin{cases} 
Q(1) & \text{if } n = 1 \\
2 + 2Q(n/4) & \text{if } n > 1 
\end{cases} \]

\[ Q(n) = O(\sqrt{n}) \]
Running Time

\[ Q(n) = \begin{cases} 
Q(1) & \text{if } n = 1 \\
2 + 2Q(n/4) & \text{if } n > 1 
\end{cases} \]

2 out of 4 regions represented by grandchild nodes

\[ Q(n) = O(\sqrt{n}) \]
Running Time

\[ Q(n) = \begin{cases} 
Q(1) & \text{if } n = 1 \\
2 + 2 Q(n/4) & \text{if } n > 1
\end{cases} \]

2 out of 4 regions represented by grandchild nodes

\[ Q(n) = O(\sqrt{n}) \]
Generalization to Higher Dimensions

- At the root, split into two subsets based on $x_1$ coordinate.
- At depth 1, partition based on $x_2$ coordinate.
  - : 
- At depth $d - 1$, partition based on $x_d$ coordinate.
- At depth $d$, partition based on $x_1$ coordinate.
  - : 

Recursion stops when the subset has one point.
$d$-Dimensional Kd-Tree

Binary tree with $n$ leaves (points)

- $O(n)$ storage
- $O(n \log n)$ construction time
- $O(n^{1-\frac{1}{d}} + k)$ query time
$d$-Dimensional Kd-Tree

Binary tree with $n$ leaves (points)

- $O(n)$ storage
- $O(n \log n)$ construction time
- $O(n^{1 - \frac{1}{d}} + k)$ query time

# reported points